MASTER EN SCIENCES INFORMATIQUES

Approche d'une classe de didacticiels

Effinier, Sébastien

Award date:
1992

Awarding institution:
Universite de Namur

Facultés Universitaires N.D. de la Paix
Namur
Institut d'informatique

Approche d'une classe de didacticiels

Mémoire présenté pour l'obtention du grade
de licencié et maître en informatique
par
Sébastien EFFINIER

Promoteur :
Claude Cherton

Je tiens à remercier mon promoteur Mr. Cherton pour ses conseils et sa disponibilité, sans lesquels ce mémoire n'aurait pu aboutir. J'aimerais également remercier les autres personnes qui ont pris le temps de lire mon mémoire et de me donner leur avis.

Résumé

Ce mémoire aborde l'étude d'une classe de didacticiel de type ICAL (Intelligent Computer Aided Learning). Le type de matière servant de base à ce didacticiel est une matière comprenant des règles, fonctions de transformation d'expressions. Le but pédagogique recherché est d'apprendre à l'élève à choisir et à appliquer ces règles. Cela est fait en lui proposant des problèmes et en essayant de lui procurer une assistance pédagogique pertinente. Pour y parvenir, il faut pouvoir générer des problèmes, posséder une certaine connaissance de la matière, essayer d'évaluer le niveau de l'élève, pouvoir analyser ses entrées et avoir une certaine stratégie pédagogique. Ces différents éléments sont regroupés dans cinq modules pour essayer d'obtenir un didacticiel cohérent tout en isolant les difficultés.

Après la définition de cette découpe et la mise en évidence de quelques aspects importants dans les différents modules, j'ébauche un peu le côté implémentation. Je ne pousse pas très loin cette seconde partie et je suis en tout cas loin d'une implémentation effective.

Abstract

This memoir starts the study of a class of ICAL (Intelligent Computer Aided Learning) tutoring systems. The basic domain of these tutoring systems is a domain containing rules, expression transformation functions. The pedagogic goal is to teach students to choose and to apply these rules. This is achieved proposing problems and trying to provide an adequate pedagogical assistance. For that, one must be able to generate problems, to evaluate the student level, to analyze his inputs and to observe a pedagogical strategy. These different elements are gathered in five modules to try to obtain a consistent tutoring system and to isolate the difficulties.

After the definition of that structure and the enlightenment of some important aspects in the different modules, I outline a little the implementation. I don't go very far in that second part and however I'm far from an effective implementation.

Table des matières

Introduction

Jusqu'à aujourd'hui, on a principalement développé trois grands types de didacticiels : les cours programmés, les simulateurs et les didacticiels de type ICAL (Intelligent Computer Aided Learning).

Les cours programmés

Les cours programmés sont nés sous forme de livres : le lecteur doit résoudre un problème et selon la réponse, ou plutôt le type de réponse qu'il fournit, le livre lui indique la page où se trouve la suite de la leçon. L'introduction de l'ordinateur avait pour objet de faciliter le travail de l'élève en le branchant automatiquement sur la suite de la leçon correspondant à sa réponse. Il faut souligner que, dans le cas du livre, c'est le lecteur lui-même qui interprète sa réponse et qui décide de quel type elle est. Dans le cas de l'ordinateur, c'est le système qui doit interpréter la réponse. Le lecteur sachant le raisonnement qu'il a suivi pour parvenir à sa réponse, l'interprétation lui est assez facile, quasi inconsciente. Le système, en revanche, aura beaucoup de difficultés pour faire cette interprétation et dans certains cas cela lui sera même impossible. De plus, tous les types de réactions possibles de l'élève devront être prévus dès la conception de ce système. Il existe un cas pour lequel le système analysera facilement l'information, c'est celui du questionnaire à choix multiple.

Ces cours sont efficaces dans de nombreuses situations, par exemple quand il s'agit d'apprendre à reconnaître des animaux selon certains critères particuliers, à connaître la généalogie d'une dynastie, etc... En général, ce sont les cas où l'on désire apprendre la structure d'une connaissance sans chercher à comprendre le pourquoi de cette structure.

Cependant, pour beaucoup d'autres apprentissages, le nombre de types de réactions est beaucoup trop élevé pour que l'on puisse envisager d'utiliser ce genre de didacticiels.

Pour faciliter la conception de tels systèmes, en vue de permettre aux enseignants d'écrire leurs propres cours programmés, on a introduit les langages-auteur qui sont des générateurs de cours programmés.

Les simulateurs

Le principe des simulateurs est tout différent. L'utilisateur est plongé dans la modélisation d'un monde réel dont on veut lui faire sentir les lois. En lui montrant la conséquence de ses actions, on lui permet de découvrir quelles sont les actions qu'il doit accomplir pour parvenir à une situation particulière. Pour pouvoir modéliser, il faut isoler un nombre limité de paramètres tout en gardant une modélisation assez proche du réel afin de rester efficace.

L'exemple le plus connu est celui du simulateur de vol dans lequel l'apprenti pilote est placé pratiquement dans les mêmes conditions que celles du monde réel et ce sans courir de risque ni pour lui-même ni pour l'avion. Il peut ainsi apprendre à appliquer la théorie qu'on lui a enseignée, par exemple les techniques de décollage et d'atterrissage, avant d'être confronté à la réalité.

Les didacticiels de type ICAL

La troisième approche est celle de l'intelligence artificielle. Celle-ci se rapproche de la méthode des cours programmés en ce sens que les deux types de didacticiels proposent un problème à l'élève et, selon sa réponse, adoptent certaines attitudes. Par contre, le cas de l'intelligence artificielle diffère sur trois points importants que je développe ci-dessous, à savoir la façon de voir l'élève, la matière enseignée et la structuration de la leçon.

Dans le cas des cours programmés, on détermine le niveau de connaissance atteint par l'élève de façon implicite, sans en garder de trace. C'est-à-dire que le niveau de l'élève découle de la réponse au problème qu'il est en train de résoudre. S'il résout correctement ce problème, son niveau s'améliore et on lui propose le problème de niveau supérieur.

Sinon, on lui propose le problème de niveau inférieur. Dans le cas de l'intelligence artificielle, on essaye de modéliser l'élève et ses connaissances, de quantifier le niveau qu'il a atteint, de reconstituer sa façon de raisonner. Par la suite, toutes ces données sont ainsi accessibles pour le système aussi bien que pour le professeur qui suit son élève.

La matière sur laquelle on travaille est également implicite dans le cas des cours programmés, elle est contenue dans la structure même du programme, ce qui fait que l'on ne peut la manipuler en tant que telle. C'est comme un programme écrit dans un langage procédural qui calcule la fonction sinus. Ce qu'il calcule est périodique mais on ne retrouve pas cette propriété dans le code du programme. Dans le cas de l'intelligence artificielle, on exprime la connaissance explicitement de telle sorte que le programme puisse y avoir accès.

Enfin, l'intelligence artificielle tente de modéliser la prise de décision, c'est-à-dire de déterminer quels sont les éléments qui font que l'on adopte telle attitude plutôt que telle autre.

On peut donc dire que l'approche de l'intelligence artificielle est caractérisée par une tentative de modélisation, et donc de compréhension, du concept d'enseignement et du concept de connaissance. Elle est destinée à essayer de pallier aux faiblesses que les autres types de didacticiels ont montrées dans certains cas.

Ajoutons que tout didacticiel, aussi bon soit-il, ne remplacera jamais un enseignant. En effet, l'enseignement est une chose extrêmement complexe faisant intervenir de nombreux facteurs concernant, bien sûr, les connaissances de l'élève mais également son état d'esprit, son caractère, etc... On ne pourra jamais intégrer dans un didacticiel la totalité des éléments qui influencent l'enseignement. Donc, un didacticiel restera toujours une aide à l'enseignement et ne remplacera jamais les enseignants.

Mon approche

Mon approche concerne une classe de didacticiels caractérisée par un but pédagogique et non par une matière. Pour des raisons qui seront développées dans le premier chapitre, cette classe de didacticiels nécessite que le système ait une certaine connaissance de la matière et de l'élève. Ceci implique que ma démarche soit du type

ICAL. Cependant, comme il peut être assez difficile de réaliser une telle étude sans pouvoir se référer à des situations concrètes d'apprentissage, j'ai parfois considéré un didacticiel particulier, celui des manipulations de fractions.

Je terminerai cette introduction en donnant les grandes lignes de mon travail.

Dans le chapitre 1, je commence par donner les caractéristiques que doivent posséder les didacticiels que j'envisage. Pour répondre à ces caractéristiques, je propose une découpe en cinq modules. Les chapitres 2 à 5 développent ces modules.

D'abord, dans le chapitre 2, j'aborde succinctement les interfaces, je propose des idées pour pouvoir générer des problèmes (module de génération) et j'essaie de formaliser le niveau de connaissance atteint par l'élève (module historique).

Ensuite, dans le chapitre 3, je discute de la structure et de la construction de la base de connaissance du système (module de connaissance).

Dans le chapitre 4, je propose un algorithme d'analyse des entrées de l'élève (module d'analyse).

Enfin, dans le chapitre 5, j'essaie de définir la notion de stratégie pédagogique et d'orienter la réflexion en vue de l'implémenter (module pédagogique).

Dans le dernier chapitre, j'aborde l'implémentation. Il ne s'agit que de quelques éléments ponctuels qui ne sont qu'ébauchés. Ils servent à donner une idée des problèmes qu'il peut y avoir.

Chapitre 1 : Caractéristiques et structure du didacticiel

Type de matières concernées

Quand on définit un didacticiel, deux choses sont déterminantes pour qu'il soit efficace :

- le type de matières sur lequel on travaille; en effet, on n'utilisera évidemment pas la même méthode pour apprendre à conduire une voiture que pour apprendre à manipuler des fractions
- l'objectif pédagogique que l'on se donne, c'est-à-dire le type de connaissance de la matière que l'on veut faire acquérir à l'élève.

Voyons d'abord le type de matières concernées par ce mémoire. Il s'agit de matières qui sont constituées d'un ensemble de règles. Une règle est une fonction qui transforme une expression en une expression équivalente. Cette notion d'équivalence est à définir dans chaque cas de figure d'après le domaine de définition des règles. Dans le cas des expressions fractionnaires numériques, la notion d'équivalence associée est l'égalité du point de vue de la valeur numérique.

Dans ce type de matières, un problème consiste en l'application de ces règles à une situation initiale pour la transformer par étapes successives en une autre situation, équivalente mais possédant un certain nombre de caractéristiques. Un problème est défini autant par ces caractéristiques que par la situation initiale. Notons que pour la notion d'application d'une règle, la définition d'une règle donnée ci-dessus ne suffit plus. En effet, on peut souvent appliquer une même règle à différentes parties d'une expression donnée. Par exemple pour l'expression (2/4)+(4/6), on peut simplifier par 2 la fraction 2/4 ainsi que la fraction 4/6. Pour définir de façon univoque le résultat de l'application d'une

règle à une expression, il faut donc donner, en plus de la règle que l'on emploie, les éléments de l'expression qui participent à l'application de cette règle. C'est ce qu'on appelle les instanciations de la règle.

On trouve de nombreux exemples de telles matières en mathématiques. Par exemple pour la dérivation de fonctions, un problème serait de trouver l'expression de la fonction équivalente à f'(x) (la situation initial) qui ne contiendrait plus de signe de dérivation. Mais les mathématiques ne sont pas le seul domaine intéressant. Le diagnostic médical est un exemple dans un tout autre domaine. Dans ce cas, un problème est de trouver la maladie correspondant aux symptômes que montrent le patient.

Mais c'est surtout l'objectif pédagogique qui est important pour déterminer le type de stratégies pédagogiques qui convient. En effet, on n'apprend pas de la même façon à connaître par coeur l'énoncé de règles et à savoir les utiliser correctement. Ce que je veux faire apprendre ici, c'est à choisir et à appliquer les différentes règles pour passer de la situation initiale à la situation finale.

Caractéristiques du didacticiel

Tout d'abord, pour atteindre mon objectif pédagogique, il faut essayer de reconstituer le plus fidèlement possible le raisonnement suivi par l'élève. Cela servira à évaluer le niveau de connaissance atteint par l'élève. En effet, le système doit avoir une connaissance du niveau atteint par l'élève pour pouvoir le guider efficacement et lui proposer des problèmes de son niveau. Je choisis que cette connaissance ne se base que sur les exercices déjà résolus, s'il y en a. Je n'envisage pas la possibilité de questions telles que "Connais-tu telle règle ?" ou "Quelles règles sont-elles applicables à tel énoncé ?", etc... Ces questions peuvent être très utiles pour évaluer le niveau de l'élève mais leur gestion est plus complexe et je n'envisage leur introduction que dans une étape ultérieure. S'il n'y a pas encore eu de problème résolu, on peut envisager un questionnaire préliminaire qui déterminera le niveau de l'élève. Ce questionnaire peut être limité à la question "Es-tu d'un niveau débutant, moyen ou bon ?". Reconstituer le raisonnement de l'élève servira aussi et surtout à l'aider efficacement quand il est bloqué ou quand il fait des erreurs.

C'est pourquoi le didacticiel que je vais envisager est un système ayant une certaine connaissance de la matière qu'il enseigne. Mais étant donné mon objectif, cette connaissance ne peut pas comprendre l'ensemble des règles dont le système pourrait avoir besoin à tout moment. En effet, comme je l'ai dit, le but recherché est d'apprendre à l'élève à choisir et à appliquer les règles. Ce but est atteint en lui soumettant des problèmes et en essayant de lui apporter un soutien pédagogique pour les résoudre. Pour pouvoir guider l'élève efficacement, il faut pouvoir "penser" comme lui. Or, l'être humain utilise beaucoup plus de règles que celles de l'ensemble minimal nécessaire à la résolution. Il utilise très souvent des cas particuliers. Mais surtout, il lui arrive d'inventer des règles, d'utiliser des astuces. S'il s'agissait uniquement pour le système de résoudre les problèmes, il serait concevable, pour des matières relativement simples, que l'ensemble minimal de règles nécessaires à la résolution soit totalement connu par le système. Mais, dans mon cas, il n'est pas possible que le système connaisse toutes les règles que l'élève pourrait utiliser. Dans ce sens, l'ensemble de base des règles sera donc incomplet. De plus, il serait intéressant qu'il ne soit pas fixé une fois pour toutes mais qu'il puisse évoluer pour s'adapter aux règles utilisées régulièrement par l'élève.

Pour pouvoir expliquer à l'élève pourquoi il s'est trompé, il serait également intéressant de pouvoir disposer d'un ensemble de règles incorrectes. Le système devra essayer de voir si une règle est cohérente ou non avec l'ensemble des règles dont il dispose. Mais si en plus de dire à l'élève qu'il utilise une règle incorrecte, le système peut lui donner le mécanisme qui lui a fait construire cette règle, l'efficacité pédagogique sera nettement accrue.

Comme je l'ai dit un peu plus haut, le système doit être capable de générer des problèmes d'une complexité donnée et nécessitant l'utilisation de règles données pour sa résolution, puisqu'il va faire évoluer l'élève en lui soumettant des problèmes. Dans le cas où un problème lui est fourni par le concepteur ou le professeur, il pourra éventuellement être accompagné de certaines informations pour aider le système à le résoudre, la connaissance du système pouvant ne pas être complète (au sens mathématique du terme), ou pour lui permettre de donner des indications plus précises à l'élève.

Une autre caractéristique essentielle est que le système doit être capable d'analyser tous les éléments de dialogue provenant de l'élève tout au long de la leçon. Dans ce dialogue, l'élève aussi bien que le système peuvent avoir l'initiative. Les actions de

l'élève, c'est-à-dire les éléments du dialogue dont l'élève a l'initiative, peuvent être soit des commandes, soit des demandes. Un exemple de commande est celle d'arrêt de la leçon. En ce qui concerne les demandes, l'élève pourrait solliciter des explications sur la résolution d'un exercice ou demander une indication parce qu'il est bloqué ou encore soumettre au système un problème qu'il n'arrive pas à résoudre. Le système peut de son côté décider de donner à l'élève une explication ou une indication. Celle-ci peut être donnée sous la forme d'un sous-problème plus simple qui, on l'espère, aidera l'élève à voir son erreur ou à voir la direction dans laquelle il doit se diriger. Venant de l'élève, il y a également les réponses aux sollicitations du système. Ces réponses peuvent être des expressions, étapes successives de la résolution d'un problème. Elles peuvent également être des énoncés de règles ou les sous-expressions auxquelles ces règles ont été appliquées, dont le système aurait besoin pour reconstituer une des étapes de la résolution de l'élève. Le système doit pouvoir classer toutes les entrées de l'élève et en extraire les informations dont il pourrait avoir besoin pour pouvoir assister l'élève. De plus, il doit baser son analyse sur sa connaissance de la matière et sur l'historique de l'élève pour être pédagogiquement pertinent.

Enfin, je veux permettre l'utilisation de stratégies pédagogiques différentes. En effet, chaque professeur possède sa propre vision de la pédagogie. Fixer la stratégie du système une fois pour toutes obligerait les professeurs à se plier à cette pédagogie, ce qui diminuerait très fort l'intérêt qu'ils pourraient trouver dans ce didacticiel. Cette stratégie doit donc être explicite dans le système, c'est-à-dire qu'un programmeur doit pouvoir y accéder, l'adapter dans une certaine mesure aux préférences du professeur, sans devoir modifier la structure du programme. Dans un avenir plus lointain, il serait même intéressant de doter le système d'un mécanisme permettant au professseur de modifier la stratégie pédagogique lui-même. C'est elle qui détermine la complexité et les règles à faire intervenir dans le problème à soumettre à l'élève. Elle gère également le dialogue avec l'élève durant la résolution d'un problème.

Structure du didacticiel

Je vais donner au didacticiel une structure modulaire afin, principalement, d'isoler les difficultés. Cette découpe en modules va apporter les avantages habituels de la découpe en modules d'une application. Mais dans mon cas, c'est surtout la possibilité de

développer les modules séparément qui m'intéresse. Ce serait beaucoup trop complexe d'aborder ce système en un seul tenant, chaque partie prise isolément étant déjà très complexe.

Pour justifier la découpe en modules, j'utiliserai principalement les critères suivants :

- cohérence temporelle, c'est-à-dire regroupement des fonctions qui sont utilisées en-même temps
- cohérence logique, c'est-à-dire regroupement des fonctions de même caractère, par exemple les fonctions mathématiques ou les fonctions graphiques
- cohérence orientée objet, c'est-à-dire regroupement des fonctions qui agissent sur les mêmes données.

De plus, lorsque deux modules sont en relation d'utilisateur-utilisé, c'est-à-dire quand le premier module fait appel aux fonctions du second sans que le second ne fasse appel aux fonctions du premier, alors les deux modules sont de niveaux différents, ce qui est un argument supplémentaire en faveur de la décomposition.

Passons maintenant en revue les différentes parties du didacticiel.

La génération des problèmes

Le système a besoin de disposer de problèmes faisant intervenir certaines règles avec une complexité donnée. Pour cela, il n'a besoin que de connaître les règles de la matière à enseigner et d'une procédure qui permet de construire un problème à partir de ces règles. On peut isoler cette procédure à condition de lui donner un accès à la base de connaissance. On obtient ainsi un module cohérent d'un point de vue temporel, utilisé uniquement lors de la génération des problèmes, et d'un point de vue logique, regroupant ce qui concerne la génération des problèmes. Ce module est utilisateur de celui qui contient la base de connaissance et est utilisé par le module pédagogique comme on le verra plus loin. Il sera appelé module de génération. On peut ainsi le développer séparément et le limiter lors de l'implémentation à une base de données contenant des problèmes avec leur complexité respective. Dans ce cas, la génération se limite à un accès à la base de données et il n'y a pas besoin d'accès à la base de connaissance.

L'historique de l'élève

Les éléments concernant le niveau de l'élève peuvent être isolés, donnant une cohérence logique et orientée objet. Le module obtenu est utilisé, comme on le verra plus loin, par le module d'analyse et le module pédagogique, et n'est pas utilisateur. Il sera appelé module historique. Ici aussi, cette découpe permet d'implémenter un système sans devoir beaucoup développer l'aspect historique au début. Mais il ne faut quand même pas l'ignorer totalement, car un didacticiel qui ne garderait aucune trace de ce que l'élève à déjà fait manquerait évidemment d'efficacité. En effet, le fait de donner à l'élève des exercices trop difficiles ou trop faciles risque de le lasser rapidement.

La pédagogie

Comme je l'ai dit au début du chapitre, c'est surtout l'objectif pédagogique que l'on s'est fixé qui détermine si une stratégie pédagogique sera efficace ou non, plutôt que la matière en elle-même. Ainsi, pour des matières différentes mais que l'on aborde avec le même objectif pédagogique, on pourra utiliser les mêmes stratégies pédagogiques. Par exemple, on pourra utiliser les mêmes méthodes pour faire apprendre la structure des différentes familles et espèces animales ou la généalogie d'une dynastie. D'autre part, on peut utiliser des stratégies pédagogiques différentes pour une même matière selon ses conceptions de la pédagogie et selon le caractère de l'élève. Si on a des objectifs pédagogiques différents, on utilisera certainement des stratégies pédagogiques différentes. On voit ainsi qu'il y a une certaine indépendance entre la matière que l'on veut enseigner et la stratégie pédagogique que l'on utilise. Je peux donc séparer ce qui est pédagogique de ce qui est connaissance de la matière : cela donne le module pédagogique et le module de connaissance. J'obtiens ainsi une cohérence logique. De plus, le module pédagogique est utilisateur du module de connaissance. En faisant cette séparation, je veux qu'un même système puisse servir pour des matières différentes mais que l'on veut aborder avec le même objectif pédagogique. De plus, cela permet à un programmeur d'adapter dans une certaine mesure la stratégie pédagogique aux préférences des professeurs qui se servent du système.

L'analyse des entrées de l'élève

Enfin, l'analyse des réponses de l'élève est une fonction à part entière qui utilise la connaissance et l'historique. C'est une couche ajoutée à ces deux aspects qui permet au module pédagogique d'en faire un meilleur usage. Elle n'a pas du tout besoin de la génération de problèmes. Elle est indépendante de la pédagogie, même s'il est vrai que les possibilités pédagogiques sont limitées par les informations que peut lui fournir l'analyse puisque le module pédagogique est utilisateur de l'analyse. Elle n'est qu'utilisatrice de l'historique et de la connaissance. C'est pourquoi j'en fais un module séparé (le module d'analyse), obtenant ainsi une cohérence logique.

En conclusion, on a 5 modules :

- un module historique
- un module de génération des problèmes
- un module de connaissance
- un module d'analyse des entrées de l'élève
- un module pédagogique

Il y a également un certain nombre d'interfaces entre ces modules et le monde extérieur, c'est-à-dire l'élève et le professeur. Ces interfaces sont un aspect très important du système. Je les développerai un peu plus dans le chapitre suivant.

Cette structure permet bien au didacticiel de posséder les caractéristiques citées au début du chapitre. Je vais maintenant expliciter un peu plus les interactions entre les différents modules.

Interactions entre les modules

Le module pédagogique utilise les interfaces pour communiquer avec l'élève et le professeur. Il prend ses décisions à l'aide du module historique et de la stratégie pédagogique que lui a définie le concepteur. Il dispose des problèmes dont il a besoin grâce au module de génération des problèmes. Il va chercher les éléments de connaissance qu'il veut mettre dans son dialogue avec l'élève dans le module de

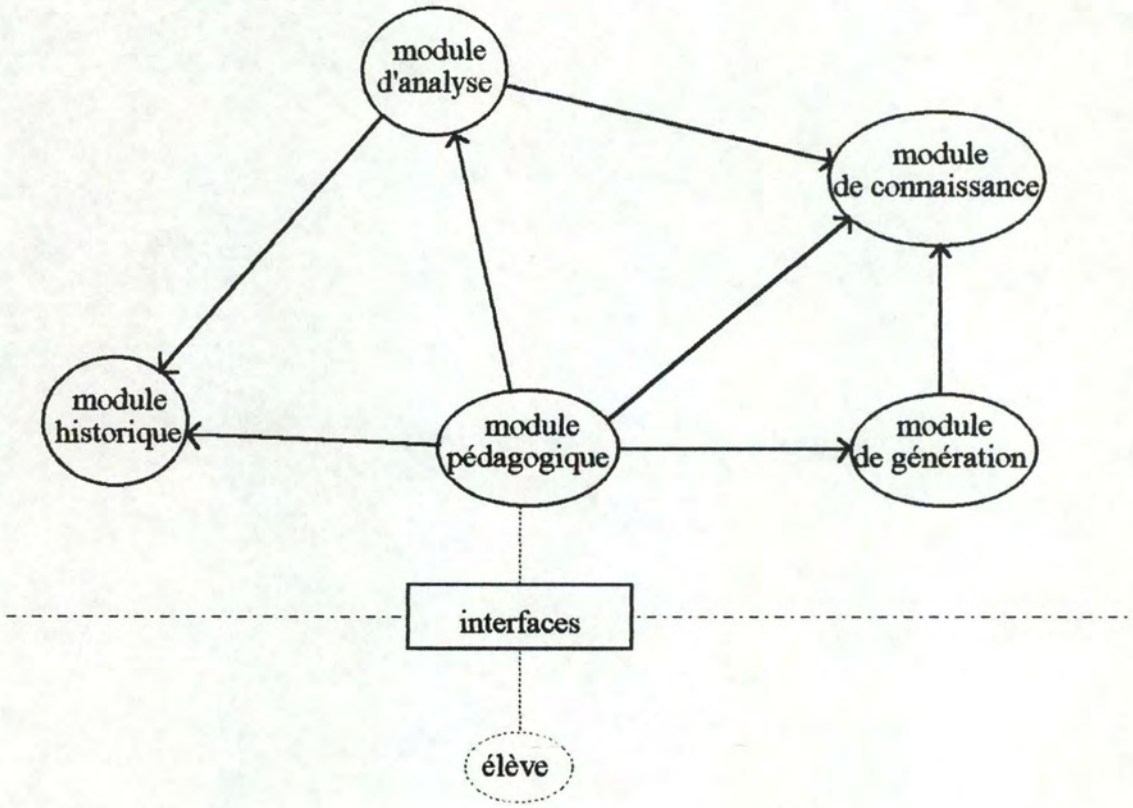
connaissance. Il peut par exemple avoir besoin de l'énoncé d'une règle. Enfin, il extrait des informations des entrées de l'élève par le biais du module d'analyse des entrées.

Le module d'analyse reçoit donc des entrées du module pédagogique et lui rend les résultats de son analyse. De plus, il base ses analyses sur la connaissance de la matière du module de connaissance et sur celle du niveau de l'élève fournie par le module historique.

Signalons enfin que le module de génération des problèmes peut se baser sur le module de connaissance, comme il sera expliqué dans le chapitre suivant.

On voit donc que les deux modules centraux d'un point de vue pédagogique sont le module pédagogique et le module d'analyse des entrées de l'élève. En effet, c'est le module pédagogique qui prend les décisions et c'est le module d'analyse qui lui permet de les prendre.

L'ensemble des modules et de leurs interactions donne le schéma de la page suivante, les flèches représentant les relations utilisateur-utilisé.



Le professeur n'apparaît pas dans le schéma parce que, bien qu'il ait accès au didacticiel, ses interventions se placent dans un contexte tout à fait différent.

Chapitre 2 : Les interfaces, la génération des problèmes et l'historique

Les interfaces

Les interfaces utilisateurs sont un aspect très important du didacticiel. Il est d'abord plus agréable de travailler avec une interface ergonomique, aussi bien pour l'élève que pour le professeur. De plus, les interfaces peuvent être un facteur pédagogiquement déterminant pour les exercices proposés à l'élève. En effet, un même problème peut être présenté de différentes façons, et ce avec des impacts pédagogiques très différents. Par exemple, si on présente la multiplication comme une loi interne et partout définie sur les réels ayant telle ou telle propriété ou si on la présente à l'aide de surfaces de rectangles, cela n'aura pas du tout le même impact. Je ne vais cependant pas développer l'aspect interface dans le cadre de mon mémoire car je considère le développement des autres modules comme prioritaire. Des interfaces sophistiquées peuvent n'être introduites que lorsque le reste du système sera opérationnel.

La génération des problèmes

L'utilité du module de génération est de fournir au système des problèmes faisant intervenir des règles données et cela avec des complexités données. Ce module peut également fournir diverses informations sur ces problèmes telles que des indications ou des astuces à utiliser pour parvenir à la solution, voire donner la solution elle-même.

Pour la génération des problèmes, plusieurs solutions sont envisageables. On peut d'abord considérer une base de données. Elle contiendrait un certain nombre de

problèmes pour chaque ensemble de règles et chaque complexité voulus. A chaque problème, on peut associer les informations dont j'ai parlé plus haut. Mais cette base de données peut être assez longue à concevoir. De plus, cette solution est très limitative au niveau de la complexité et de la combinaison des règles intervenant dans un exercice donné. Seuls sont possibles les exercices prévus lors de la conception du système. Une solution est de permettre au professeur d'ajouter des problèmes à une base de données prédéfinie. Mais alors, cela devient assez pénible pour le professeur car il est obligé de penser en terme de cas particuliers, de problèmes totalement instanciés. Or un professeur a plutôt l'habitude de penser d'abord en terme de type de problème. Il ne pense en terme de problème instancié que lorsqu'il propose vraiment l'exercice.

Pour se rapprocher de la façon de penser d'un professeur et ce, sans beaucoup de difficultés supplémentaires, on peut concevoir des schémas de problèmes. Chaque schéma nécessite l'utilisation d'un certain nombre de règles et contient un certain nombre de variables. Quand on a choisi les règles que l'on veut faire utiliser par l'élève, on prend le schéma correspondant et on instancie les variables. D'abord, s'il n'y a pas de schéma faisant intervenir exactement les règles voulues, on peut soit essayer d'en combiner, soit prendre celui qui correspond le mieux. Cette approximation du schéma exact est à définir dans l'algorithme de recherche du schéma. Ce peut être un schéma faisant intervenir plus de règles comme ce peut en être un qui en fait intervenir moins. Ensuite, pour les instanciations, ce peut être par des nombres ou bien par des sous-expressions. On peut ainsi faire varier la difficulté de l'exercice en changeant le type des instanciations. L'exercice sera plus simple si les instanciations sont réalisées par des petits nombres que si elles le sont par des sous-expressions. On n'est donc plus limité au niveau de la complexité mais seulement au niveau des combinaisons de règles.

Rappelons que les schémas intéressants sont ceux qui font intervenir les règles que l'on veut faire apprendre à l'élève. De plus, un schéma correspond en fait à une règle à l'envers, c'est-à-dire qu'au lieu d'aller d'un énoncé vers une solution, on va d'une solution vers un énoncé. Ces deux éléments montrent que la génération des schémas est étroitement liée à la génération des règles. Je développerai dans le prochain chapitre la génération des règles.

C'est ainsi que ces schémas peuvent être générés dynamiquement par le système en fonction des règles ou fixés lors de la conception du système. Le premier cas est intéressant si l'ensemble des règles est fixe, c'est-à-dire si le système ne génère pas automatiquement de nouvelles règles. En effet, on peut alors être sûr de la pertinence de

toutes les règles. En revanche, si le système génère des règles, il est plus prudent que ce soit une personne extérieure qui fixe ces schémas ou qui détermine la pertinence des règles nouvellement générées, ce qui revient au même. Cela peut évidemment être une combinaison des deux : par exemple, le système génère des schémas qui doivent être validés par une personne extérieure avant de pouvoir être utilisés.

Enfin, la génération des problèmes pourrait être faite à partir des règles elles-mêmes. Ce dernier cas est similaire à la génération automatique des schémas, mis à part le fait qu'au lieu d'être explicites, les schémas sont implicitement générés chaque fois qu'un problème doit être généré. Ici, il est obligatoire que la validation par une personne extérieure se fasse au niveau des règles puisque l'on n'a pas accès aux schémas eux-mêmes.

Complexité d'un exercice

La notion de complexité d'un exercice est associée à la génération des problèmes. Cette complexité est évidemment liée aux règles que l'exercice fait intervenir lors de sa résolution. Définissons d'abord la complexité d'une règle.

Si l'on veut avoir une réelle efficacité pédagogique, la complexité d'une règle doit être décomposée en plusieurs éléments. En effet, il est courant qu'un élève connaisse l'énoncé des règles mais qu'une fois qu'il est devant un problème à résoudre, il soit complètement perdu, qu'il ne voie pas du tout quelles règles il peut appliquer. Il est courant également qu'un élève connaisse l'énoncé et sache appliquer correctement une règle mais que, lors de la résolution d'un exercice, il ne sache pas laquelle des règles applicables il doit utiliser pour arriver à la solution. Il arrive souvent aussi qu'un élève essaye d'appliquer une règle qu'il connaît et qu'il se trompe malgré tout lors de son application. Il y a ainsi quatre aspects différents :

- la difficulté à mémoriser la règle (mémorisation)
- la difficulté à reconnaître que l'on peut l'appliquer (applicabilité)
- la difficulté à l'appliquer (application)
- la difficulté à reconnaître que son utilisation mène effectivement à la solution (pertinence).

Parmi ces quatre aspects différents, seuls les trois derniers dépendent de l'expression à laquelle la règle est appliquée, autrement dit des instanciations faites lors de l'application de la règle.

Notons enfin que l'on cherche à apprendre ces règles à l'élève et non à les classer par ordre de complexité. C'est pourquoi chacun de ces nombres doit être dédoublé. Ainsi à chaque règle il faut associer, en plus de la complexité intrinsèque (celle qui reste quand on connaît parfaitement la règle) une complexité pour un élève donné. Un élève qui ne connaît pas bien une règle la trouvera plus complexe qu'un élève qui la connaît très bien. En revanche, tous les élèves seront d'accord pour dire que telle règle est plus complexe que telle autre une fois qu'ils les connaissent toutes les deux suffisamment bien. Cette complexité pour un élève donné est en fait une indication du niveau que l'élève a atteint. Si on prend la complexité pour l'élève comme un pourcentage de la complexité intrinsèque, on obtient le niveau de l'élève sur une échelle de cent : une valeur de 100 correspond à une connaissance parfaite, c'est-à-dire que la complexité pour l'élève est la même que la complexité intrinsèque et une valeur de 0 correspond à une connaissance nulle, c'est-à-dire que la complexité intrinsèque est infinie par rapport à la complexité pour l'élève.

Je définis la complexité d'un exercice, ou plutôt les complexités, comme la somme des complexités des règles différentes concernées. Pour le calcul de la complexité globale d'un exercice, je ne compterai que l'occurrence pour laquelle l'instanciation est la plus complexe si une même règle intervient plusieurs fois dans la résolution. En effet, le fait de devoir appliquer deux fois la même règle ne double pas la complexité de l'exercice. Si on a su l'appliquer une fois correctement, il n'est pas difficile de l'appliquer une seconde fois. Par contre, cela augmente la quantité de travail à fournir. Mais je ne développerai pas cette notion. Je noterai simplement que, dans un exercice, il faut essayer de maximiser une certaine utilité. Celle-ci croît en même temps que la complexité, du moins jusqu'à un certain point. Il ne faut en effet pas proposer un exercice que l'élève ne saurait pas résoudre. Et elle décroît lorsque le travail croît, du moins à partir d'un certain point, car il faut quand même driller un minimum l'élève. On voit donc que cette notion d'utilité est très complexe et qu'elle relève de la pédagogie.

L'historique

Le module historique fournit les éléments d'évaluation du niveau de connaissance atteint par l'élève, ce qui permet au système de donner à l'élève une aide pertinente.

Comme je l'ai dit dans le premier chapitre, le système ne se base que sur les problèmes déjà résolus pour évaluer le niveau de l'élève. On ne peut évidemment pas retenir tous les détails de toutes les résolutions mais on peut en garder quelques uns et faire la synthèse du reste. Je choisis de ne garder des détails que pour la leçon courante et de faire la synthèse sous la forme de nombres. L'historique se décompose donc en deux parties : un ensemble d'éléments à long terme, basés sur la résolution des exercices des leçons précédentes aussi bien que de ceux de la leçon courante, et un ensemble d'éléments à court terme ne se basant que sur la leçon courante.

Le long terme comprend l'ensemble des nombres représentant le niveau atteint par l'élève (la complexité pour l'élève dont j'ai parlé un peu plus haut). Rappelons que, comme pour la complexité d'un problème, le niveau de l'élève est relatif à chacune des règles et, pour chacune d'elles, il se décompose en quatre éléments : la mémorisation, l'applicabilité, l'application et la pertinence. Ici aussi les trois derniers éléments dépendent des instanciations. Rappelons également que ce niveau atteint est exprimé en pour-cent.

Tous ces nombres doivent être mis à jour lors de la résolution des exercices. L'utilisation correcte d'une règle augmente les nombres relatifs à cette règle. Cette augmentation dépend du nombre de fois que l'élève a utilisé correctement la règle les dernières fois qu'il l'a utilisée. En effet, après un certain nombre d'utilisations correctes d'une règle, on peut considérer qu'elle est parfaitement connue, même si au début de son apprentissage l'élève ne l'utilisait jamais correctement. De même, l'utilisation incorrecte diminue ces nombres et cette diminution dépend du nombre d'utilisations incorrectes lors des utilisations les plus récentes de la règle. Le lien exact entre les utilisations correctes ou incorrectes et le niveau de l'élève est un problème important. C'est en effet lui qui déterminera si les exercices proposés sont du niveau de l'élève ou non. C'est également un problème complexe qui mérite une étude approfondie.

Le long terme sert à la détermination de la complexité et des règles à faire intervenir dans le problème à proposer à l'élève. Il aide aussi à l'analyse des réponses de l'élève et donc à la prise de décision pédagogique. Si le système fait l'hypothèse que la réponse de l'élève comprend l'utilisation d'une règle qu'il n'est pas censé connaître, le système peut lui demander l'énoncé de la règle qu'il a utilisée pour voir s'il la connaît effectivement ou bien s'il a trouvé cette réponse en utilisant une règle incorrecte. Par contre, si l'élève ne trouve pas une solution qui ne fait intervenir que des règles qu'il est censé bien connaître, le système devra peut-être revoir ce niveau de connaissance et revenir en arrière dans son enseignement.

Le court terme sert à raffiner les éléments du long terme et également à personnaliser autant que possible le dialogue avec l'élève. Parmi les composantes du court terme, il y a le nombre d'utilisations correctes ou incorrectes de chaque règle avec les instanciations correspondantes. Il permet par exemple de dire des choses du genre "Tu l'as pourtant bien utilisée dans tel exercice" ou bien "Tu as encore oublié tel élément".

La mise à jour de l'historique est un problème délicat. En effet, il ne faut pas faire des mise à jours trop fréquentes car cela risque de faire osciller très fort le système et de le ralentir inutilement. Mais il ne faut pas non plus les faire trop tard sinon l'historique ne correspondra plus à la réalité.

Chapitre 3 : La connaissance de la matière

Le module de connaissance sert à :

- donner au module d'analyse les moyens de reconstituer le raisonnement suivi par l'élève
- donner au module pédagogique les éléments dont il pourrait avoir besoin lors de ses explications à l'élève, par exemple l'énoncé d'une règle ou ses cas de pertinence
- donner au module de génération des problèmes les connaissances dont il pourrait avoir besoin pour générer ses problèmes.

Je ne développerai pas le dernier point qui n'est à considérer que dans le cas où la génération des problèmes se fait à partir de schémas ou directement à partir des règles et non dans le cas d'une base de données.

En outre, je reparlerai du second point dans le chapitre relatif à la pédagogie, le chapitre 5.

Développons le premier point.

Le système doit pouvoir retrouver le chemin suivi par l'élève lors de son raisonnement ou, au moins, essayer de le reconstituer, en faisant un certain nombre d'hypothèses. Il doit donc être capable de suivre plus ou moins le même type de raisonnement et pour cela, il doit disposer d'un ensemble de règles ressemblant autant que possible à celles dont dispose l'élève. Mais comme je l'ai déjà dit, il n'est de toute façon pas possible que le système connaisse toutes les règles que l'élève pourrait utiliser. Si l'élève utilise une règle que le système ne connaît pas, ce dernier doit être capable de se rendre compte qu'il ne connaît pas cette règle.

On acceptera que le système se trompe à certains niveaux. Par exemple, lorsqu'il fait une hypothèse sur le chemin suivi par l'élève, il est très possible qu'il se trompe, et il doit pouvoir remettre cette hypothèse en question. En effet, si le système s'obstine dans son erreur, il risque de perdre beaucoup d'efficacité.

Par contre, il y a des points sur lesquels on n'acceptera pas que le système se trompe. Ainsi, s'il lui arrive de considérer correcte une règle ou une réponse incorrecte, il n'est pas fiable et l'élève et surtout le professeur refusera de l'utiliser. Mais cela ne veut pas dire que le système ne peut pas laisser l'élève s'engager dans une voie sans issue ou utiliser une règle incorrecte. En effet, cela peut être pédagogiquement très intéressant de laisser l'élève découvrir seul ses erreurs.

Comme je l'ai déjà dit au chapitre 1, pour être pédagogiquement efficace, il ne faut pas seulement que le système puisse dire si une règle est correcte ou non. Il doit également pouvoir dire à l'élève pourquoi c'est faux, c'est-à-dire comment il a pu construire une règle incorrecte. Pour ce faire, le système peut avoir à sa disposition soit un ensemble des règles incorrectes les plus courantes, soit un générateur de fausses règles.

La connaissance correcte peut être divisée en deux parties : les règles et les méta-règles. Les règles permettent de modifier réellement l'expression pour essayer de lui faire acquérir les caractéristiques nécessaires. Elles modifient une expression valide en une autre expression valide. Les méta-règles, quant à elles, sont des fonctions d'évaluation de la pertinence des règles. Elles ne transforment pas une expression valide en une autre expression valide comme le font les règles mais elles donnent une indication sur la règle à utiliser pour arriver à la situation finale. C'est-à-dire que les méta-règles sont des heuristiques.

Les règles et leurs structures

Il est intéressant d'associer aux règles un certain nombre de relations sous-jacentes qui structurent les règles en réseaux (ou en arbres, un réseau pouvant être représenté par un arbre si on répète les noeuds qui sont destination de plusieurs arcs).

Parmi ces relations, il y a la particularisation. Deux règles seront reliées par un arc lorsque le noeud d'arrivée est égal au noeud de départ dans lequel on a remplacé certaines variables par des valeurs particulières et que l'on a effectué les simplifications que ces instanciations permettaient. Ainsi, une règle peut donner lieu à plusieurs règles particulières. Considérons par exemple, la règle générale pour la somme de deux fractions : (a/b)+(c/d)=(ad+cb)/bd. Un cas particulier est obtenu lorsque d est multiple de b : (a/b)+(c/mb)=(am+c)/mb. Cette instanciation a permis de simplifier le résultat par b. Un autre cas particulier est obtenu lorsque b=d : (a/b)+(c/b)=(a+c)/b. Ici aussi, on a pu simplifier par b. De même, une règle particulière peut provenir de la particularisation de plusieurs règles plus générales. Par exemple, le cas de la règle de la somme de deux fractions où b=d est un cas particulier de la règle générale mais aussi du cas où d est un multiple de b. Donc, la particularisation et la généralisation, qui est la relation sous-jacente inverse, donnent lieu toutes les deux à des réseaux plutôt qu'à des arbres. Ces deux relations sont intéressantes parce que l'être humain raisonne souvent en terme de cas particuliers et le fait de connaître explicitement cette relation permet au système de donner à l'élève des explications pédagogiquement plus pertinentes.

Une autre relation est la relation de prérequis. Une règle est reliée à une autre si la connaissance de la première est jugée nécessaire à l'enseignement de la seconde. Dans ce cas également cela donne lieu à un réseau plutôt qu'à un arbre. Cette relation est très intéressante. En effet, c'est ce réseau qui va guider l'évolution de l'enseignement. Avant de pouvoir aborder l'apprentissage d'une règle, il faut que toutes les règles prérequises soient suffisamment connues.

L'ensemble de base des règles

Rappelons d'abord qu'à chaque règle est associé un ensemble de nombres représentant les différentes complexités de la règle ainsi qu'un ensemble de nombres représentant le niveau de connaissance de l'élève. Tout ce qui est relatif à ces ensembles de nombres a été développé dans la partie relative à la complexité d'un exercice et dans celle relative à l'historique au chapitre précédent.

En ce qui concerne la constitution de l'ensemble de règles, il y a plusieurs solutions. On peut par exemple fixer cet ensemble une fois pour toutes lors de la

conception du système. Mais cela demande une réflexion très importante lors de cette conception. Car le fait d'oublier une règle peut enlever au système une bonne partie de son efficacité pédagogique. Il faut en tout cas doter le système d'une procédure qui essaye de déterminer si une règle est cohérente avec sa base de connaissance ou non. Dans cette procédure, il ne faut pas oublier qu'il y a des règles pour lesquelles il ne sera pas possible de déterminer si elles sont ou non cohérentes avec la base de connaissance du système, même si cela se présente assez rarement.

On peut aussi permettre au système de générer automatiquement de nouvelles règles. Dans ce cas, il faut faire très attention à ce que l'on génère : il faut essayer de ne générer que des règles pertinentes, c'est-à-dire des règles qui seront utilisées par la suite assez fréquemment par le système. La difficulté réside dans la formalisation de la notion de pertinence.

On peut concevoir un générateur qui mémorise toutes les successions de règles utilisées lors de la résolution des exercices, et qui crée de nouvelles règles en composant les règles qui se succèdent souvent. Mais ce système possède quelques faiblesses. D'abord, après combien d'occurrences peut-on dire qu'une succession de règles est pertinente ? Ensuite, lorsque l'on a généré une nouvelle règle, il faut lui associer les nombres représentant sa complexité. Comment déterminer ces nombres ? Ce n'est en tout cas pas en additionnant les complexités des règles qui composent cette nouvelle règle. Cette dernière est souvent plus complexe que les règles qui la constituent mais demande moins de travail (voir le chapitre 2 pour la notion de travail). Enfin, pour pouvoir guider efficacement l'élève, il faudrait déterminer dans quels cas il est pertinent d'appliquer la nouvelle règle.

Une autre façon de générer de nouvelles règles serait d'essayer de détecter certaines instanciations particulières des différentes règles, par exemple les instanciations faisant intervenir des neutres ou des absorbants d'opérateurs, et en créant de nouvelles règles faisant intervenir les simplifications dues à l'utilisation de ces valeurs particulières. Dans ce cas, il faudra également leur associer des cas de pertinence et des nombres représentant leurs complexités.

Si l'on veut que le système dispose de fausses règles, on peut, comme pour les règles correctes, soit les fixer au départ, soit concevoir un générateur. Ici aussi, les fixer au départ risque d'être moins efficace mais les générer est très délicat. Car il faut dans ce cas-ci que le concepteur essaye de relever les mécanismes que suivent les élèves pour

construire les mauvaises règles et qu'il implémente ces mécanismes. Par exemple, il y a la généralisation incorrecte. De la règle (a/b)*(c/d)=(a*c)/(b*d), ils déduisent parfois (a/b)+(c/d)=(a+c)/(b+d).

Les méta-règles

Les méta-règles correspondent à une certaine heuristique. Elles nous servent à simuler le mécanisme humain qui fait que l'on choisit telle règle plutôt qu'une autre pour essayer de s'approcher de la situation finale.

Comme pour les règles, il est intéressant de leur associer des structures sous-jacentes en arbre ou en réseau telles que la particularisation ou le prérequis et de leur associer des nombres de complexité et des niveaux de connaissance de l'élève. La prise en compte de ces méta-règles apporte énormément au niveau pédagogique mais, étant un essai de formalisation du pourquoi du comportement humain, c'est encore beaucoup plus complexe que l'essai de formalisation du comportement lui-même. C'est pourquoi je n'analyserai pas plus loin cet aspect dans le cadre de mon mémoire.

Chapitre 4 : L'analyse des entrées de l'élève

Voyons un peu plus précisément ce que l'on est amené à analyser au cours d'une leçon et les informations que l'on doit en extraire.

La première chose que l'élève rentre, c'est son nom. Cette donnée ne nécessite pas d'analyse car c'est la réponse à une question qui n'attend qu'un seul type de réponse et la seule information qui nous intéresse est le ou les fichiers associés à cet élève. Dans le cas de la première leçon de cet élève, j'ai envisagé la possibilité qu'il introduise une indication de son niveau de connaissance pour faire une initialisation de son historique. Ici aussi, cette donnée ne nécessite pas d'analyse.

Lorsque l'élève veut arrêter la leçon (ce qu'il peut faire à tout moment), il introduit la commande de fin de leçon. Cette commande pourrait être une demande, si le système ne permet pas que l'on s'arrête au milieu de la résolution d'un problème. Une autre demande de l'élève pourrait être la demande d'une indication pour l'aider à progresser dans la résolution. Les demandes et les commandes nécessitent une analyse car elles peuvent survenir à tout moment et le système doit pouvoir les distinguer des solutions proposées par l'élève ou des autres entrées possibles. La seule information qu'il faut en extraire est la commande ou la demande qui correspond.

Considérons maintenant le cas où le système propose un problème à l'élève. Supposons que l'élève sache le résoudre, partiellement ou entièrement, et qu'il donne une expression correcte (par expression correcte, j'entends déduite de l'énoncé par application correcte de règles correctes). D'une part, le système doit pouvoir se rendre compte qu'il s'agit d'une expression, et non d'une demande ou d'une commande. D'autre part, il doit

essayer de déterminer si elle est correcte. S'il y parvient, il aura besoin des informations suivantes pour continuer la leçon :

- l'entrée est une expression
- cette expression est correcte
- le chemin que l'on suppose que l'élève a suivi est un tel
- toutes les règles de ce chemin sont bien connues par l'élève ou telle et telle règle est mal connue par l'élève selon l'historique que l'on dispose sur lui, c'est-à-dire que la réponse qu'il a donnée est ou non plausible.

Si la réponse est jugée non plausible, le système doit essayer de déterminer si son historique correspond bien au niveau de l'élève, cela étant du ressort du module pédagogique.

Si la réponse est jugée plausible et que l'élève continue à donner des réponses correctes et plausibles jusqu'à la réponse finale, le système peut passer à l'exercice suivant, sans oublier de mettre à jour l'historique, ceci aussi étant du ressort du module pédagogique. C'est évidemment le cas le plus simple.

Supposons maintenant que l'élève donne une réponse incorrecte. D'abord, le système doit ici aussi se rendre compte que c'est une expression. Ensuite, après avoir vainement essayé de déterminer qu'elle était correcte, il doit essayer de déterminer qu'elle est incorrecte. Dans le cas des fractions numériques, cela peut être fait en évaluant numériquement l'expression. Si elle a une valeur différente de celle de l'énoncé, alors on peut dire qu'elle est incorrecte. Il doit alors essayer d'expliquer pourquoi elle est incorrecte, c'est-à-dire de trouver la ou les règles que l'élève a mal utilisées. Cela sera fait à l'aide des fausses règles ou du générateur de fausses règles. Les informations qu'il a extraites sont alors :

- l'entrée est une expression
- cette expression est incorrecte
- selon les hypothèses faites sur son raisonnement, l'élève a utilisé telle et telle fausses règles
- les explications pédagogiques associées à ces règles sont les suivantes.

Si le système ne parvient pas à identifier les fausses règles que l'élève a pu employer ou s'il ne parvient pas à déterminer si l'expression est correcte ou non, il peut demander à l'élève des indications sur le chemin qu'il a suivi. Ces indications peuvent être l'énoncé d'une règle qu'il a utilisée, les instanciations avec lesquelles il a appliqué

cette règle ou une expression intermédiaire située entre l'énoncé et la réponse que le système ne comprend pas. Le choix du type d'indication demandée à l'élève dépend de la stratégie pédagogique, et donc du module pédagogique. Mais, le module d'analyse doit pouvoir analyser ces indications. En ce qui concerne l'énoncé d'une règle, il doit essayer de déterminer si elle est cohérente ou non avec son ensemble de base de connaissances. Il peut ensuite reprendre son analyse de l'entrée précédente avec cette donnée supplémentaire pour le guider. Pour les instanciations, il doit vérifier si elles peuvent bien correspondre à la règle donnée par l'élève. Ici également, il peut reprendre son analyse avec cette donnée supplémentaire pour le guider. Le cas de l'expression intermédiaire peut être traité de la même façon que si l'élève avait directement proposé cette expression comme une solution et non comme une indication.

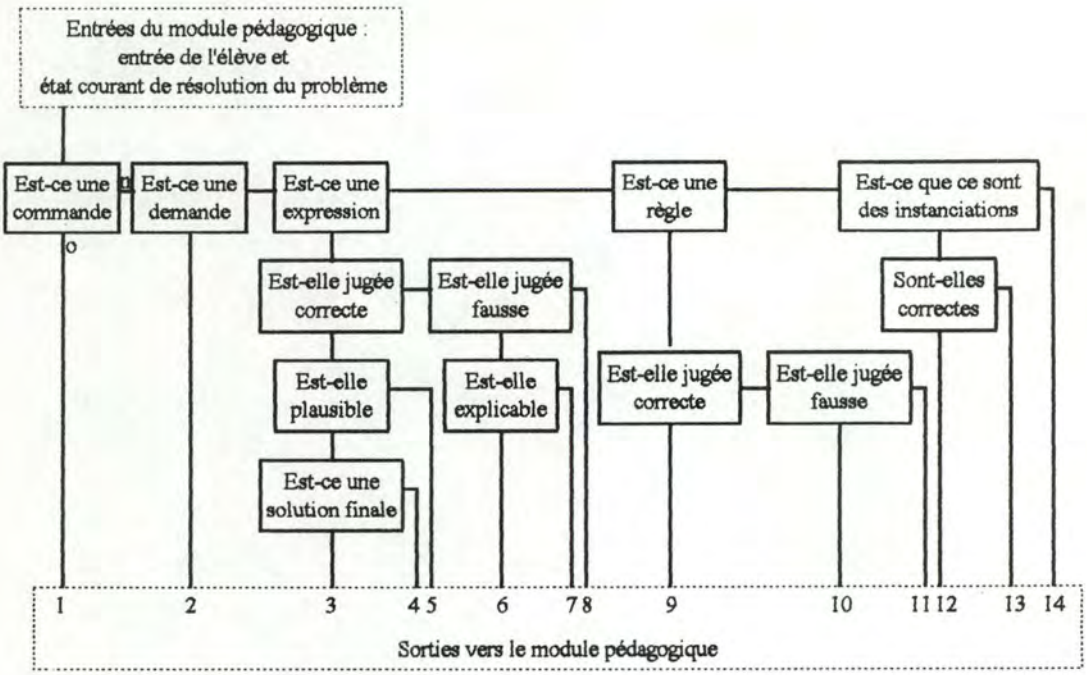
Enfin, dans le cas où l'élève fait une faute de frappe ou introduit une entrée non prévue par le système, ce dernier doit signaler à l'élève une erreur de syntaxe. Dans un système plus sophistiqué, on pourrait essayer de déterminer le type dont l'entrée se rapproche le plus, ce pourquoi elle s'en écarte, voire une proposition de correction.

En résumé, il y a 5 types d'entrée à analyser :

- les commandes
- les demandes
- les expressions, étapes successives de la résolution
- les énoncés de règles
- les instanciations.

De plus, ces entrées ne sont pas analysées de façon isolée mais dans un certain contexte. J'appellerai ce contexte l'état courant de résolution du problème. Il peut contenir, entre autres, l'état de l'arbre de résolution déjà calculé par le système, les étapes déjà parcourues pour la résolution du problème courant, les règles utilisées correctement et celles utilisées incorrectement lors de la résolution du problème courant et les indications, telles que l'énoncé d'une règle ou des instanciations, fournies par l'élève pour essayer de reconstituer le raisonnement menant à l'étape de résolution courante.

L'algorithme suivant permet les comportements décrits ci-dessus.



Les informations fournies par le module d'analyse au module pédagogique pour chaque sortie comprennent les éléments suivant, en plus de l'état courant de résolution du problème :

1-nom de la commande

2-nom de la demande

3-indication de la fin de la résolution de l'exercice avec une éventuelle explication d'une solution alternative plus simple

4-indication d'une solution intermédiaire correcte et plausible

5-indication de la non-plausibilité d'une expression correcte

6-expression incorrecte avec des explications

7-expression incorrecte sans explication

8-expression dont on ne sait pas si elle est correcte ou incorrecte

9-règle jugée correcte

10-règle jugée incorrecte avec éventuellement des explications

11-règle non évaluable, jugée ni correcte ni incorrecte

12-acceptation des instanciations avec le résultat de leur utilisation

13-indication d'une erreur d'instanciation avec la ou les instanciations fautives

14-indication d'une erreur de syntaxe avec éventuellement une proposition de correction.

Voyons maintenant les interactions entre le module d'analyse et les autres modules.

Interactions avec les autres modules

Le module d'analyse utilise le module de connaissance principalement pour essayer de reconstituer le raisonnement suivi par l'élève. Il lui demande également de faire la distinction entre une expression, un énoncé de règle et des instanciations. Il s'en sert pour essayer de déterminer si une règle est correcte ou non. Enfin, il s'en sert pour tester la validité d'instanciations.

Le module d'analyse se base sur le module historique pour tester la plausibilité des règles qu'il suppose que l'élève a utilisées. Il peut également s'en servir pour se guider lorsqu'il essaye de reconstituer le raisonnement suivi par l'élève, par exemple en essayant d'appliquer d'abord les règles bien connues par l'élève.

Enfin, le module d'analyse est utilisé par le module pédagogique. Cela a déjà été discuté et cela le sera encore dans le chapitre suivant.

Chapitre 5 : La pédagogie

Le module pédagogique est le coordinateur du didacticiel. C'est lui qui décide des règles à faire intervenir, et avec quel degré de complexité, dans les problèmes à soumettre à l'élève. Pour cela, il suit la stratégie pédagogique que lui a définie le concepteur. Cette stratégie doit tenir compte de l'historique de l'élève. Le module pédagogique utilise le module de génération des problèmes. Il prend les entrées de l'élève et les donne à analyser au module d'analyse. Il décide de l'évolution de la leçon en fonction des résultats de cette analyse et ici aussi en suivant sa stratégie pédagogique.

Il y a donc 2 aspects importants dans le module pédagogique : la stratégie pédagogique et les interactions avec les autres modules.

La stratégie pédagogique

Qu'est-ce qu'une stratégie pédagogique ?

La stratégie pédagogique est l'ensemble des attitudes à adopter et la façon de les enchaîner dans les diverses situations pour guider le mieux possible l'élève. Elle dépend évidemment de l'objectif pédagogique que l'on s'est fixé. Elle dépend de la vision du professeur en matière de pédagogie. Elle dépend aussi de l'élève, de son caractère, de son humeur, du niveau qu'il a atteint, etc... On sera moins exigeant si on sait que l'élève relève d'une longue maladie. Le type d'aide sera différent avec un élève très doué qu'avec un peu doué. Il y a ainsi un grand nombre de facteurs dont il faudrait tenir compte pour être pédagogiquement efficace.

Tenir compte de tout ce qui peut influencer l'élève n'est pas possible. Un professeur humain tient compte de beaucoup d'éléments de manière souvent inconsciente. Mais il y a également des attitudes de l'élève qu'il ne peut expliquer. Pourquoi l'élève est-il plus attentif un jour et totalement dissipé un autre ? Pourquoi butte-t-il toujours sur cette règle qu'on lui a déjà expliquée des dizaines de fois ?

Pour un didacticiel, c'est encore plus compliqué car il faudrait formaliser tous ces éléments avant de pouvoir les implémenter. Or, avant de pouvoir formaliser quelque chose, il faut pouvoir l'expliquer. Et il y a des choses que l'on ne peut que très difficilement expliquer, et donc formaliser. La détection de l'état de fatigue et l'évaluation du niveau de concentration de l'élève sont deux de ces éléments que l'on ne peut que très difficilement implémenter. On pourrait peut-être implémenter ces aspects à l'aide de notions telles que le temps de réflexion. Si le temps de réflexion de l'élève est beaucoup plus long que d'habitude, on peut lui demander s'il est distrait ou s'il est fatigué. Le fait qu'un élève préfère telle règle plutôt que telle autre pourtant plus efficace est un cas encore plus difficile à formaliser. Ce sont des aspects qui relèvent de la psychologie. Je ne vais pas en tenir compte car ils sont très complexes. De plus, on peut les ignorer dans un premier temps pour voir les résultats que l'on obtient. On en introduirait ensuite quelques uns pour évaluer si cela vaut la peine de les étudier plus en profondeur ou non.

Les éléments dont je tiendrai compte sont ceux que l'on peut déduire des entrées de l'élève et qui sont repris dans l'historique tel que je l'ai défini au chapitre 2. Rappelons que l'historique comprend les détails de la résolution des exercices de la leçon courante ainsi qu'un niveau atteint par l'élève pour chaque règle sous la forme d'un ensemble de nombres.

Définition d'une stratégie pédagogique

A partir de l'historique, le module pédagogique doit déterminer quelles sont les règles qu'il veut faire intervenir dans le prochain problème à soumettre à l'élève. Ces règles peuvent être déterminées à partir de la relation de prérequis que j'ai définie au chapitre 3. On divise l'arbre de la relation de prérequis en quatre parties :

- les règles déjà bien connues
- les règles que l'on peut faire apprendre, c'est-à-dire celles dont toutes les règles prérequises appartiennent au premier ensemble

- les règles du reste de l'arbre
- les règles que l'on est effectivement en train de faire apprendre, c'est un sous-ensemble de la seconde partie.

Le système doit alors choisir les règles qu'il veut faire intervenir dans le problème parmi celles du premier ensemble et surtout parmi celles du quatrième. Lorsqu'une règle est devenue assez connue, elle passe des second et quatrième ensembles au premier et les règles du troisième ensemble, qui n'avaient plus que cette règle comme prérequis mal connu, peuvent passer dans le second. A l'inverse, si le système se rend compte que l'élève ne connaît plus très bien une règle du premier ensemble, il doit faire retourner dans le second ensemble cette règle et dans le troisième toutes les règles pour lesquelles elle est prérequise, il doit reprendre l'apprentissage de ces règles. De plus, lorsque le nombre de règles constituant le quatrième ensemble tombe sous un certain seuil, il doit faire passer une ou plusieurs règles du second ensemble dans le quatrième.

Une fois que le système a déterminé les règles qu'il voulait faire intervenir dans le problème à soumettre à l'élève, il doit encore déterminer avec quelle complexité elles doivent intervenir. Cette complexité doit être proportionnelle au niveau atteint par l'élève. En effet, proposer à l'élève un exercice ne faisant intervenir que des règles qu'il connaît bien, et ce, avec une faible complexité, ne lui apprendra rien. Mais lui proposer un exercice qui ne fait intervenir que des règles qu'il ne connaît pas bien, et ce, avec une complexité importante, le découragera et il n'en retirera pas grand chose.

Pendant la résolution des problèmes, le module d'analyse des entrées de l'élève donne ses interprétations des entrées au module pédagogique. Passons en revue les attitudes que peut prendre le module pédagogique en fonction des interprétations du module d'analyse que l'on a vues au chapitre précédent.

S'il s'agit d'une expression qui est une solution finale du problème, le module pédagogique peut faire un certain nombre de choses avant de passer à l'exercice suivant. Il peut, par exemple, demander au module d'analyse si l'élève n'aurait pas pu utiliser une solution plus simple. Si c'est le cas, le module pédagogique peut l'expliquer à l'élève ou lui dire qu'il y en a une et lui demander d'essayer de la trouver. Il peut également donner un message d'encouragement ou de félicitation. Le choix des actions à effectuer avant le passage à l'exercice suivant est à définir dans la stratégie pédagogique.

S'il s'agit d'une étape intermédiaire correcte et jugée plausible, le module pédagogique attend l'étape suivante. Ce ne doit évidemment pas être une attente passive.

D'abord, il doit contrôler que l'élève ne reste pas trop longtemps sans se manifester. S'il considère qu'une réponse tarde trop à arriver, il peut demander à l'élève s'il a un problème ou s'il veut de l'aide. Ensuite, pendant l'attente, le système peut déjà rechercher certaines solutions possibles. Cela permettra d'accélérer l'analyse de la prochaine réponse de l'élève.

S'il s'agit d'une étape intermédiaire correcte mais jugée non plausible, le système peut interroger l'élève pour essayer de déterminer s'il connaît les règles pour lesquelles on n'est pas sûr de sa connaissance ou s'il a suivi un autre chemin, correct ou incorrect. Cela peut être fait, par exemple, en demandant comment il a obtenu la partie de l'expression qui est jugée non plausible.

S'il s'agit d'une étape intermédiaire incorrecte mais explicable, c'est-à-dire que le module d'analyse a pu déterminer la ou les règles incorrectes que l'élève a utilisées, le module pédagogique peut choisir entre deux attitudes extrêmes. D'un côté, il peut laisser continuer l'élève pour essayer qu'il trouve lui-même son erreur. De l'autre, il peut lui expliquer son erreur. Entre ces deux extrêmes, il peut, par exemple, dire à l'élève qu'il se trompe sans lui donner plus de précisions. C'est le type d'erreur qui peut déterminer vers quelle extrême se diriger. Si l'élève utilise une règle correcte mais qui ne le conduira pas à une solution, il peut être bon de le laisser continuer, tout en ne le laissant pas se perdre dans un chemin trop complexe. Par contre, s'il fait une erreur de distraction, il est inutile de le laisser finir l'exercice pour lui dire à la fin qu'il s'est trompé de signe au début de l'exercice.

S'il s'agit d'une étape intermédiaire incorrecte et non explicable, le module pédagogique doit dire à l'élève que sa réponse est fausse mais qu'il ne la comprend pas. Pour essayer de la comprendre, le système peut demander à l'élève une étape intermédiaire, située entre cette réponse et la dernière étape que le système avait comprise ou une règle qu'il a utilisée pour parvenir à cette expression ou encore les instanciations avec lesquelles il a utilisé cette règle. Si l'élève donne l'énoncé d'une règle ou d'une étape précédente ou des instanciations correctes, le module pédagogique doit demander au module d'analyse de reprendre l'analyse en utilisant ce nouvel élément. Par contre, si l'élève donne l'énoncé d'une règle ou des instanciations incorrectes, on se retrouve dans le cas de l'étape intermédiaire incorrecte et explicable, sauf que l'on ne dispose peut être pas d'explication sur la règle incorrecte utilisée. S'il donne une expression incorrecte, on peut considérer que l'erreur se trouve avant cette nouvelle expression. Il est évidemment possible que l'élève ait fait une autre erreur après, et un

système sophistiqué pourrait essayer de voir si ce n'est pas le cas pour mettre à jour l'historique de l'élève. Excepté pour la mise à jour de l'historique, on peut reprendre la leçon sur base de cette nouvelle expression.

S'il s'agit d'une expression pour laquelle le module d'analyse n'a pas pu déterminer si elle était correcte ou incorrecte, il doit essayer de la comprendre comme dans le cas précédent. Mais dans ce cas-ci, il ne peut même pas dire à l'élève que sa réponse est fausse, il peut seulement dire qu'il ne la comprend pas.

Enfin, s'il s'agit d'une commande ou d'une demande, le module pédagogique doit déterminer s'il l'accepte ou non et doit l'exécuter s'il l'accepte.

Lors de la résolution d'un problème, le module pédagogique peut être amené à fournir de l'aide à l'élève. Cela peut survenir à la demande de l'élève ou à l'initiative du système. Cette aide peut prendre de nombreuses formes. Le système peut donner les cas d'applicabilité d'une règle, la règle que l'élève devrait utiliser, le résultat de l'application d'une règle, un sous-problème faisant intervenir les mêmes règles mais plus simplement, etc...

Rappelons enfin que dans tous les cas, le module pédagogique doit mettre à jour l'historique.

Le professeur et la stratégie pédagogique

Dans la structure de didacticiel que je propose, j'ai isolé un module pédagogique pour permettre au professeur de modifier lui-même la stratégie pédagogique du système dans un avenir plus ou moins lointain.

La première idée qui vient à l'esprit d'un informaticien est d'écrire différentes procédures correspondant aux différentes réactions possibles, c'est-à-dire les résultats d'analyse possibles, et à l'historique de l'élève. Cette découpe permettrait au professeur de réécrire une procédure qui ne lui conviendrait pas. Mais cette solution demande au professeur d'être informaticien ou de disposer d'un informaticien, de connaître le langage dans lequel a été écrit le didacticiel et de disposer d'une solide documentation sur les différentes procédures.

Pour pouvoir envisager de façon réaliste que le professeur modifie lui-même la stratégie pédagogique, il faudrait définir un langage de description de cette stratégie pédagogique. Pour définir ce langage, il faut tenir compte des éléments suivants :

- le langage doit être adéquat pour décrire une stratégie pédagogique, cela veut dire que ce langage sera sans doute assez complexe étant donnée l'objectif visé
- le langage doit essayer de tenir compte des concepts utilisés habituellement en pédagogie pour essayer de faciliter l'apprentissage des professeurs.

Mais les concepts à dégager pour pouvoir définir ce langage sont très complexes. C'est pourquoi, on n'obtiendra peut-être pas un langage permettant au professeur de définir sa stratégie avec toutes les nuances qu'il pourrait désirer. J'espère pourtant que l'on arrivera à donner au professeur plus de liberté que la possibilité de jouer sur quelques paramètres. De toute façon, quand ce langage aura été mis au point, les professeurs auront besoin d'un certain temps d'apprentissage avant de pouvoir l'utiliser.

Un système disposant d'un tel langage possèderait toute la souplesse voulue tout en donnant au professeur la possibilité d'utiliser cette souplesse assez facilement.

Les interactions avec les autres modules

La génération des problèmes

Le module pédagogique donne au module de génération des problèmes les règles qu'il a sélectionnées avec leurs complexités respectives. Le module de génération construit un problème répondant à ces spécifications, au moyen d'une base de données ou de schémas comme c'est décrit au chapitre 2. En plus de l'énoncé de ce problème, il peut passer au module pédagogique des éléments pédagogiquement utiles tels que la solution du problème ou des astuces à utiliser pour y parvenir.

La richesse de l'interface avec le module pédagogique, c'est-à-dire la variété des informations que ce dernier peut demander au module de génération des problèmes, détermine la richesse du comportement pédagogique que peut adopter le module pédagogique. Cela est vrai également pour les autres modules, comme on va le voir dans la suite.

L'historique

Le module pédagogique accède à l'historique à trois moments de l'exécution :

- lors de l'initialisation des données relatives à un nouvel élève
- lors de la détermination des spécifications auxquelles doit répondre le problème à soumettre à l'élève
- lors de la mise à jour de l'historique au cours de la résolution d'un exercice.

L'accès à l'historique pour déterminer les spécifications auxquelles doit répondre le problème à soumettre à l'élève est principalement un accès en lecture à l'historique à long terme. Le module pédagogique donne au module historique le numéro des règles et le module historique lui retourne les nombres représentant le niveau de l'élève pour ces règles. Il peut aussi y avoir accès au court terme de l'historique pour raffiner le choix des règles comme, par exemple, ne pas donner trop d'exercices du même type consécutivement.

L'accès lors de l'analyse de la réponse est aussi un accès en lecture au long terme de l'historique. Il s'agit de savoir si toutes les règles que l'on suppose avoir été utilisées par l'élève sont assez connues par celui-ci que pour qu'il les ait correctement utilisées. Ici aussi, cela peut être raffiné par le court terme. Par exemple, s'il ne connaissait pas bien une règle mais qu'il l'a correctement utilisée lors de la leçon courante, on peut considérer que c'est plausible qu'il l'ait encore employée correctement.

La détermination des moments précis de mise à jour de l'historique est un problème délicat, comme je l'ai dit au chapitre 2. Cette mise à jour nécessite un accès en lecture et écriture. Il faut d'abord mettre à jour les éléments à long terme : les nouvelles valeurs des nombres représentant le niveau de l'élève pour une règle sont calculées en fonction des anciennes valeurs et des utilisations correctes ou incorrectes lors des dernières utilisations. Il faut également mettre à jour les éléments à court terme tels que ces nombres d'utilisations correctes ou incorrectes.

Le module de connaissance

Le module pédagogique peut vouloir accéder au module de connaissance pour formuler les explications qu'il veut donner à l'élève. Cela peut être simplement pour avoir

l'énoncé d'une règle, comme ce peut être pour avoir la règle générale correspondant à une règle particulière ou les conditions de pertinence d'une règle.

Ici, comme pour le module de génération de problème, la richesse de ce que peut demander le module pédagogique détermine la richesse de l'attitude pédagogique qu'il peut adopter.

Le module d'analyse des entrées de l'élève

Les interactions entre le module d'analyse et le module pédagogique ont été développées tout au long des deux derniers chapitres. Résumons le processus d'échange. D'un part, le module pédagogique donne au module d'analyse une entrée de l'élève et l'état courant de la résolution du problème. L'entrée de l'élève peut être une expression, un énoncé de règle, des instanciations, une commande, une demande ou une combinaison de tout cela. D'autre part, le module d'analyse donne comme résultat le type de l'entrée ainsi que les éléments, qui dépendent de ce type, dont le module pédagogique a besoin pour faire son choix de l'attitude à adopter.

Plus encore ici que pour les autres modules, la richesse des informations que peut fournir le module d'analyse détermine la richesse des attitudes que peut adopter le module pédagogique.

Chapitre 6 : Regard sur l'implémentation

Dans ce chapitre, j'aborde quelques problèmes d'implémentation relatifs à chacun des modules. Le langage de programmation que j'emploie est le Prolog. J'utilise ce langage parce qu'il est déclaratif, c'est-à-dire que le code du programme est sous la forme de règles (les prédicats) qu'un moteur d'inférence interprète. Le programme peut ainsi analyser son propre code, déduire des propriétés sur les résultats qu'il produit. C'est un avantage considérable pour ma base de connaissance. En effet, un des points important pour l'efficacité du didacticiel est le fait que le système puisse évoluer selon les règles utilisées par l'élève (voir le chapitre 3).

Un autre avantage de Prolog est que le code est facilement découpable en modules et facilement extensible. C'est donc un très bon langage pour le prototypage et la mise au point d'un logiciel. Cet avantage est lié également au fait que Prolog est un langage déclaratif.

Ces avantages ne sont pas sans une contrepartie. En effet, Prolog étant un langage déclaratif, il nécessite une analyse du programme très différente de celle à laquelle nous sommes habitués avec les langages procéduraux tels que Pascal. Cette nouvelle façon de penser demande un certain temps d'apprentissage et d'adaptation. De plus, c'est un langage d'exécution plus lente. Lorsque le didacticiel sera au point, il faudra probablement envisager de le réécrire dans un langage plus rapide.

La version exacte de Prolog que j'ai utilisée est le MacProlog, un Prolog pour Macintosh. Les caractéristiques particulières à cette version n'apparaissent que rarement dans les parties que j'aborde.

Le module historique

Rappelons que l'historique est divisé en des éléments à court terme et des éléments à long terme. Voyons d'abord le long terme.

Le long terme

Le long terme est constitué de 4 nombres représentant les niveaux de l'élève pour chaque règle (voir le chapitre 2). Cela peut être facilement réalisé par un fichier texte possédant une ligne par règle. Chaque ligne contient le numéro de la règle suivie des 4 nombres. Quatre opérations sont à effectuer sur ce fichier.

1) Ouvrir le fichier et l'initialiser pour un nouvel élève.

Quelques prédicats sont propres au MacProlog :

- dvol(Y), donne le numéro du répertoire courant
- find_file(Nom,X), donne le numéro du répertoire correspondant au fichier Nom
- open(Nom), ouvre le fichier Nom du répertoire courant
- create(Nom), crée un nouveau fichier dans le répertoire courant

Le code suivant regarde s'il y a déjà un fichier pour l'élève dans le répertoire courant. S'il y en a un, il l'ouvre, sinon il en crée un nouveau et l'initialise. L'initialisation est faite au moyen d'un vecteur de valeurs défini par le concepteur.

```
ouvre(Nom) :- dvol(Y),find_file(Nom,X),X=Y,!,open(Nom).
ouvre(Nom) :-
    create(Nom),
    init(Nom).
```

2) Lire les niveaux de l'élève correspondant à une règle

Les prédicats propres au MacProlog sont :

- recall(histoire,Nom), correspond à une variable globale, Nom prend la valeur qui a été associée à histoire lors d'un précédent appel à remember(histoire,Nom)
- seek(Nom,0), initialise la lecture du fichier à la position 0
- fr(Nom,Format,Donnees), lit dans le fichier Nom les données Donnees selon le format Format.

Le code suivant recherche séquentiellement la ligne correspondant à la règle Num et met dans C les niveaux correspondants.

```
lire(Num,C) :-
    recall(histoire,Nom),
    seek(Nom,0),
    fr(Nom,[q(3),'-',q(3),',',q(3),',',q(3),'~M'],[Num2|C2]),
    cherche(Nom,Num,Num2,C,C2).

/* cherche(Nom,Num1,Num2,C1,C2) cherche dans le fichier Nom la règle Num1 à partir
de la règle Num2 et met dans C1 les complexités correspondantes; C2 contient les
complexités correspondant à la règle Num2 */
cherche(Nom,Num,Num,C,C) :- !.
cherche(Nom,Num,_,C,_) :-
    fr(Nom,[q(3),'-',q(3),',',q(3),',',q(3),'~M'],[Num2|C2]),
    cherche(Nom,Num,Num2,C2,C).
```

3) Ecrire les niveaux de l'élève correspondant à une règle

L'écriture se fait par une recherche séquentielle de la ligne à modifier suivie de l'écriture des nombres contenus dans C.

Un prédicat est propre à MacProlog :

- fw(Nom,Format,Donnéees), écrit dans le fichier Nom les données Donnees selon le format Format.

```
ecrire(Num,C) :-
    recall(histoire,Nom),
    seek(Nom,0),
    fr(Nom,[q(3),'-'],Num2),
    recherche(Nom,Num,Num2,C).

/* recherche(Nom,Num1,Num2,C) cherche dans le fichier Nom la règle Num1 à partir de
la règle Num2 et remplace les complexités correspondantes par les valeurs contenues
dans C */
recherche(Nom,Num,Num,C) :- fw(Nom,[q(3),',',q(3),',',q(3)],C),!.
recherche(Nom,Num,_,C) :-
    fr(Nom,[q(3),',',q(3),',',q(3),'~M',q(3),'-'],[_,_,_,Num2]),
    recherche(Nom,Num,Num2,C).
```

4) Fermer le fichier

C'est une opération très simple qui se résume à :

```
ferme :- recall(histoire,Nom),close(Nom).
```

Le court terme

Le court terme reprend les dernières utilisations de chaque règle. On peut les stocker au moyen de 4 listes par règle. Ces listes reprennent la nature, correcte ou incorrecte, des dernières utilisations de la règle pour les 4 complexités différentes (voir le chapitre 2 pour ces complexités). Chaque élément de la liste vaudra soit 1 si l'utilisation correspondante à été correcte, soit 0 dans le cas contraire.

Il faut régulièrement faire la synthèse de ces listes pour mettre à jour le long terme. Pour cela, il faut définir une fonction associant à chaque liste une nouvelle valeur pour le long terme. Cette fonction dépend évidemment de l'ancienne valeur du long terme.

Le module de génération

Pour la génération des problèmes, j'envisagerai 2 cas :

- une base de données de problèmes
- une base de données de schémas

Dans le cas d'une base de données de problèmes, on a trois parties par problème :

- les règles à mettre en oeuvre et les complexités avec lesquelles elles seront utilisées
- l'énoncé du problème
- un ensemble d'indications dont la structure devra être étudiée plus en détail par le concepteur de la base de données

Dans ce cas-ci, il n'y a qu'un algorithme à concevoir, celui du choix du problème. S'il y a un problème qui met en oeuvre les règles que l'on désire avec les bonnes complexités, c'est assez simple. Par contre, s'il n'y en a pas, le concepteur de l'algorithme doit décider s'il prend un problème mettant en oeuvre plus, moins ou un ensemble de règles aussi proche que possible de celles désirées. Il a la même décision à prendre pour les complexités.

Dans le cas d'une base de données de schémas, chaque entrée de la base de donnée sera un prédicat de la forme :

schema(Liste_regles,Enon_deb,Compl,Enon_fin,Liste_vars).

Liste_regles est la liste des règles mises en oeuvre par le schéma, Enon_deb est l'énoncé du problème avant passage dans le schéma, Compl est la liste des complexités de mise en oeuvre des règles, Enon_fin est l'énoncé après passage dans le schéma et Liste_vars est la liste des variables qu'il reste à instancier relativement à chaque règle mise en oeuvre. Les complexités contenues dans Compl concernent surtout des complexités d'applicabilité et de pertinence. En effet, la complexité relative à la mémorisation d'une règle ne dépend pas du problème. Quant à celle relative à l'application, elle dépend des

instanciations avec lesquelles on utilise la règle plutôt que du reste du problème. C'est pourquoi elle sera déterminée par les instanciations des variables contenues dans Liste_vars.

On a ici deux algorithmes à concevoir :

- l'algorithme de choix et de combinaison des différents schémas pour obtenir le problème faisant intervenir les règles et les complexités d'applicabilité et de pertinence voulues
- l'algorithme de choix des instanciations pour obtenir les complexités d'application voulues.

Le module de connaissance

Ce module est évidemment très dépendant de la matière. Comme je l'ai dit dans l'introduction, je vais utiliser les manipulation de fraction pour pouvoir concrétiser un peu ce module.

Voyons d'abord comment représenter les expressions à l'intérieur du système. La solution qui permet le plus facilement de reconnaître une expression d'une autre donnée est de disposer de 4 constructeurs représentant les 4 opérations : div, mul, add et sub. Cette solution rend également aisée la manipulation des expressions. En effet, dans une expression telle que 3+4*2, il faut tenir compte de la priorité des opérateurs pour pouvoir l'évaluer. Par contre, add(3,mul(4,2)) est sans ambiguïté et ne nécessite pas de notion de priorité pour son évaluation.

Les règles constituent une part importante du module de connaissance. Leur rôle principal est la construction de l'arbre de résolution ou plutôt de la partie de cet arbre qui peut être utile à l'analyse des réponses de l'élève. Si on ne dispose pas de méta-règles, on ne peut, a priori, donner plus d'importance à une règle plutôt qu'à une autre et la construction de l'arbre devrait donc se faire en largeur d'abord. Mais cela explose très vite. J'ai mis en annexe une liste de règles sans méta-règle. Cette liste ne contient pas beaucoup de règles redondantes, elle ne reflète en tout cas pas la façon de raisonner de l'élève. Pourtant elle produit déjà une explosion de l'arbre de résolution après quelques niveaux. Il faut donc que la base de règles soit mise sous la forme :

regle(Num_reg,Expr_deb,Expr_fin,Instan) :- enonce_meta_reg,enonce_reg.

La construction n'étant plus aveugle, on peut l'envisager en profondeur d'abord, du moins jusqu'à une certaine profondeur. Il ne faut en effet pas se perdre dans une branche infinie. L'introduction des méta-règles évite ainsi au système de devoir manipuler un arbre de résolution d'une taille trop importante.

On a donc deux aspects importants auxquels il faut faire attention :

- le choix des règles : il faut qu'elles soient pertinentes, c'est-à-dire utiles à la reconstitution du raisonnement de l'élève, et il faut essayer de disposer des règles que l'élève utilisera le plus fréquemment et de celles que l'on veut lui faire apprendre
- la constitution des méta-règles : il faut essayer qu'elles guident le mieux possible la construction de l'arbre de résolution.

Ces méta-règles sont de deux types. Le premier type concerne la matière elle-même : dans tel cas de figure il est préférable d'utiliser telle règle plutôt que telle autre. Ce premier type de méta-règles fait partie du module de connaissance. Le second type provient du fait que l'on essaye d'expliquer et non de résoudre. Les méta-règles de ce second ensemble doivent être basées sur l'historique de l'élève, car elles doivent essayer de faire choisir d'abord les règles qui seront utilisées par l'élève et celles que l'on veut lui faire apprendre. De plus, elles doivent faire partie du module d'analyse et non du module de connaissance.

En plus des règles et des méta-règles, il y a, dans le module de connaissance, toute une série de fonctions annexes, servant, par exemple, à vérifier si une expression est une solution finale. J'en donne ici cinq exemples. Dans ces exemples, j'utilise un prédicat propre à MacProlog : map(predic,List_donnees). Ce prédicat vérifie si tous les éléments de la liste List_donnees vérifie le prédicat predic.

1-L'évaluation numérique : elle peut servir à vérifier que la réponse de l'élève n'est pas numériquement équivalente à l'énoncé et qu'elle n'est donc pas correcte.

```
/* Expressions irréductibles */
evalue(X,X) :- integer(X).
evalue(div(X,Y),div(X,Y)) :-
    pgcd(X,Y,P),Y=\=1,P=1.
```

```
/* Traitement des expressions ne possédant qu'un constructeur */
evalue(div(X,1),Y) :-
    evalue(X,Y).
evalue(div(X,Y),Z) :-
    pgcd(X,Y,P),
    X2 is X/P,Y2 is Y/P,
    evalue(div(X2,Y2),Z).
evalue(mul(X,Y),Z) :-
    map(integer,[X,Y]),Z is X*Y.
evalue(add(X,Y),Z) :-
    map(integer,[X,Y]),Z is X+Y.
evalue(sub(X,Y),Z) :-
    map(integer,[X,Y]),Z is X-Y.

/* Traitement des expressions possédant plus d'un constructeur */
evalue(div(X,Y),Z) :-
    evalue(X,X2),
    evalue(Y,Y2),
    eval_res(div(X2,Y2),Z).
evalue(mul(X,Y),Z) :-
    evalue(X,X2),
    evalue(Y,Y2),
    eval_res(mul(X2,Y2),Z).
evalue(add(X,Y),Z) :-
    evalue(X,X2),
    evalue(Y,Y2),
    eval_res(add(X2,Y2),Z).
evalue(sub(X,Y),Z) :-
    evalue(X,X2),
    evalue(Y,Y2),
    eval_res(sub(X2,Y2),Z).
```

```
/* Traitement des expressions de la forme operateur(X,Y) où opérateur est un des 4
constructeurs et X et Y sont soit des entiers soit des fractions */
eval_res(div(X,Y),Z) :-
    map(integer,[X,Y]),
    evalue(div(X,Y),Z).
eval_res(div(div(X1,Y1),X2),Z) :-
    evalue(div(X1,mul(Y1,X2)),Z).
eval_res(div(X2,div(X1,Y1)),Z) :-
    evalue(div(mul(Y1,X2),X1),Z).

eval_res(mul(X,Y),Z) :-
    map(integer,[X,Y]),
    evalue(mul(X,Y),Z).
eval_res(mul(div(X1,Y1),X2),Z) :-
    evalue(div(mul(X1,X2),Y1),Z).
eval_res(mul(X2,div(X1,Y1)),Z) :-
    evalue(div(mul(X1,X2),Y1),Z).

eval_res(add(X,Y),Z) :-
    map(integer,[X,Y]),
    evalue(add(X,Y),Z).
eval_res(add(div(X1,Y1),X2),Z) :-
    evalue(div(add(X1,mul(Y1,X2)),Y1),Z).
eval_res(add(X2,div(X1,Y1)),Z) :-
    evalue(div(add(X1,mul(Y1,X2)),Y1),Z).

eval_res(sub(X,Y),Z) :-
    map(integer,[X,Y]),
    evalue(sub(X,Y),Z).
eval_res(sub(div(X1,Y1),X2),Z) :-
    evalue(div(sub(X1,mul(Y1,X2)),Y1),Z).
eval_res(sub(X2,div(X1,Y1)),Z) :-
    evalue(div(sub(X1,mul(Y1,X2)),Y1),Z).
```

2-Le test d'irréductibilité : il sert à vérifier si une expression est une solution finale.

```
reduit(X) :- integer(X).
reduit(div(X,Y)) :- pgcd(X,Y,P),P=1,Y=\=1.
```

3-Les fonctions calculant les diviseurs communs à deux entiers : elles servent au calcul des dénominateurs communs et à la détermination des simplifications de fractions possibles.

```
/* Calcul du plus grand commun diviseur */
pgcd(X,X,X) :- !,integer(X).
pgcd(X,Y,P) :- map(integer,[X,Y]),X>Y,!,W is X-Y,pgcd(W,Y,P).
pgcd(X,Y,P) :- map(integer,[X,Y]),W is Y-X,pgcd(W,X,P).

/* Recherche d'un diviseur commun à X et à Y compris entre I et F */
divise(X,Y,I,F,I) :- I<F,map(divise(I),[X,Y]).
divise(X,Y,I,F,Z) :- I2 is I+1,I2<F,divise(X,Y,I2,F,Z).

/* Vérification de la divisibilité de Y par X */
divise(X,Y) :-
    map(integer,[X,Y]),X>1,Z is Y/X,integer(Z).
```

4-Le test de la syntaxe d'une expression : vérifie si une donnée est une expression syntaxiquement correcte. C'est normalement le rôle des interfaces utilisateur mais, puisqu'elles sont absentes, je place cette fonction dans ce module.

```
expr(X) :- integer(X).
expr(div(X,Y)) :- map(expr,[X,Y]).
expr(mul(X,Y)) :- map(expr,[X,Y]).
expr(add(X,Y)) :- map(expr,[X,Y]).
expr(sub(X,Y)) :- map(expr,[X,Y]).
```

Le module d'analyse

Rappelons que l'état courant de résolution est composé des éléments suivants :

- la partie de l'arbre de résolution déjà calculée
- les étapes déjà parcourues par l'élève lors de la résolution du problème courant
- la liste des règles correctement utilisées et celle des règles incorrectement utilisées par l'élève au cours de la leçon courante
- les indications données par l'élève pour permettre au système de reconstituer le raisonnement menant à l'étape courante (énoncé de règle ou instanciations).

L'algorithme donné au chapitre 4 se traduit de manière assez directe par le code suivant :

```
/* Traitement des commandes */
analyse(Entree,Etat,Etat,1,[Com]) :-
    commande(Entree,Com),!.
/* Traitement des demandes */
analyse(Entree,Etat,Etat,2,[Dem]) :-
    demande(Entree,Dem),!.
/* Traitement des expressions */
analyse(Entree,Etat1,Etat2,Num,Donnees_suppl) :-
    expr(Entree),!,
    anal_expr(Entree,Etat1,Etat2,Num,Donnees_suppl).
/* Traitement des règles */
analyse(Entree,Etat1,Etat2,Num,Donnees_suppl) :-
    regle(Entree),!,
    anal_regle(Entree,Etat1,Etat2,Num,Donnees_suppl).
/* Traitement des instanciations */
analyse(Entree,Etat1,Etat2,Num,Donnees_suppl) :-
    inst(Entree),!,
    anal_inst(Entree,Etat1,Etat2,Num,Donnees_suppl).
```

```
/* Traitement de l'erreur syntaxique */
analyse(Entree,Etat,Etat,14,Correct) :-
    corrige(Entree,Correct).

/* Expression jugée correcte */
anal_expr(Entree,Etat1,Etat3,Num,Donnees_suppl) :-
    expr_corr(Entree,Etat1,Etat2,Chemin),!,
    anal_corr(Entree,Etat2,Chemin,Etat3,Num,Donnees_suppl).
/* Expression jugée incorrecte */
anal_expr(Entree,Etat1,Etat3,Num,Donnees_suppl) :-
    expr_fausse(Entree,Etat1,Etat2),!,
    anal_fausse(Entree,Etat2,Etat3,Num,Donnees_suppl).
/* Expression jugée ni correcte ni incorrecte */
anal_expr(_,Etat,Etat,8,[]).

/* Expression jugée correcte et plausible */
anal_corr(Entree,Etat1,Chemin,Etat2,Num,Donnees_suppl) :-
    expr_plausible(Chemin),!,
    anal_plausible(Entree,Etat1,Chemin,Etat2,Num,Donnees_suppl).
/* Expression jugée correcte mais non plausible */
anal_corr(_,Etat,Chemin,Etat,5,Non_plausible) :-
    non_plaus(Chemin,Non_plausible).

/* Solution finale jugée correcte et plausible */
anal_plausible(Entree,Etat1,Chemin,Etat2,3,Autre_sol) :-
    finale(Entree),!,
    autre(Etat1,Chemin,Etat2,Autre_sol).
/* Solution intermédiaire jugée correcte et plausible */
anal_plausible(_,Etat,_,Etat,4,[]).

/* Expression jugée fausse et explicable */
anal_fausse(Entree,Etat1,Etat2,6,Explication) :-
    explique(Entree,Etat1,Etat2,Explication),!.
/* Expression jugée fausse et non explicable */
anal_fausse(_,Etat,Etat,7,[]).
```

```
/* Règle jugée correcte */
anal_regle(Entree,Etat1,Etat2,9,[]) :-
    regle_corr(Entree,Etat1,Etat2),!.
/* Règle jugée incorrecte */
anal_regle(Entree,Etat1,Etat2,10,Explication) :-
    regle_fausse(Entree,Etat1,Etat2,Explication),!.
/* Règle jugée ni correcte ni incorrecte */
anal_regle(_,Etat,Etat,11,[]).

/* Instanciations correctes */
anal_inst(Entree,Etat1,Etat2,12,Resul) :-
    inst_corr(Entree,Etat1,Etat2,Resul),!.
/* Instanciations incorrectes */
anal_inst(Entree,Etat1,Etat2,13,Inst_faut) :-
    inst_fausse(Entree,Etat1,Etat2,Inst_faut).
```

Parmi les prédicats qu'il reste à concevoir, certains sont assez simples. Par exemple, commande se réduit à une liste de prédicats de la forme :

```
commande(Nom_comm,Comm).
```

D'autres prédicats sont beaucoup plus compliqués. L'un d'eux est expr_corr. Pour concevoir cette fonction, il faut décider de la stratégie de recherche dans l'arbre de résolution (en largeur d'abord, en profondeur d'abord, heuristique, etc...). Dans le cas d'une recherche heuristique, il faut définir des méta-règles, celles dont j'ai parlé à la page 45. Il faut également, si on trouve plusieurs chemins possibles, choisir l'un d'eux. Il faut mettre une limite à sa recherche pour ne pas faire attendre l'élève trop longtemps (limite sur le nombre de niveaux de profondeur de recherche dans l'arbre, limite sur le nombre de branches visitées, limite sur le temps d'exécution, etc...).

Le module pédagogique

En attendant le langage de description de la pédagogie dont j'ai parlé au chapitre 5, le module pédagogique peut être conçu sous la forme d'un ensemble de procédures.

Chaque procédure traite un type d'entrées de l'élève et son entête est de la forme :

traite(Etat_deb,Num,Donnees_suppl,Etat_fin).

Etat_deb, Num et Donnees_suppl sont donnés par le module d'analyse. Plus précisément, Num et Donnees_suppl sont les résultats de l'analyse et Etat_deb est l'état courant de résolution après l'analyse et avant le traitement. Etat_fin quant à lui est l'état courant de résolution après le traitement.

La procédure correspondant à la rentrée de l'élève dans le système est un cas particulier. Il n'y a pas encore eu d'entrée de l'élève et l'état courant de résolution est vide. Comme on débute la leçon, on peut considérer que l'on vient de terminer la résolution du problème précédent, ce qui correspond à un résultat d'analyse de type 3. L'appel est donc le suivant :

traite([],3,[],Etat_fin).

Lors de cet appel, il faut :

- demander le nom de l'élève
- ouvrir ou initialiser le fichier du long terme correspondant et initialiser le court terme
- choisir et proposer le premier problème
- attendre une entrée de l'élève
- analyser cette entrée
- traiter cette entrée.

Cela donne :

```
traite([],3,[],Etat_fin) :-
    demande(Nom),
    initialise(Nom),
    choisit(Probl),
    affiche(Probl),
    attend(Entree,[],Etat1),
    analyse(Entree,Etat1,Etat2,Num,Donnees_suppl),
    traite(Etat2,Num,Donnees_suppl,Etat_fin).
```

Les algorithmes d'ouverture et d'initialisation de l'historique ainsi que celui du choix du problème ont été discutés au début du chapitre. L'affichage dépend de la version de Prolog que l'on utilise. L'attente ne doit pas être passive, comme je l'ai dit au

chapitre 5. Pendant l'attente, on peut modifier l'état courant de résolution : c'est la raison pour laquelle il y a deux paramètres en plus de l'entrée attendue de l'élève. Il s'agit de l'état de résolution avant et après l'attente. L'analyse, quant à elle, a été discutée dans la partie précédente sur le module d'analyse.

Une leçon se résume alors par le prédicat suivant :

```
lecon :- traite([],3,[],Etat_final),ferme.
```

Tous les résultats d'analyse doivent ainsi être traités, le traitement dépendant de l'état courant de résolution. La commande est le cas le plus simple. Par exemple, lorsque l'élève veut arrêter la leçon, en supposant qu'il s'agisse d'une commande et non d'une demande, cela donne :

```
traite(Etat,1,[quit],Etat) :- mise_a_jour,affiche(['Au revoir.']).
```

En effet, il n'y a qu'à mettre à jour l'historique à long terme en fonction de l'historique à court terme. De plus, l'état après traitement n'a pas d'importance étant donné que je ne l'utilise pas.

En général, tout traitement se compose de trois parties. La première partie est faites de prises de décision de la part du système, il choisit une attitude pédagogique en fonction de la dernière entrée de l'élève et de l'état courant de résolution. La seconde partie concerne l'affichage du résultat de ces prises de décision. Cela peut être l'affichage d'un message, d'une question, d'un énoncé de problème ou d'une combinaison de ces éléments. La dernière partie est l'attente d'une réaction de l'élève et son traitement, c'est-à-dire :

```
attend(Entree,Etat_deb,Etat1),
analyse(Entree,Etat1,Etat2,Num,Donnees_suppl),
traite(Etat2,Num,Donnees_suppl,Etat_fin).
```

On obtient ainsi un véritable dialogue entre l'élève et le système. Dans certains cas, il n'y a pas de troisième partie, par exemple pour la commande de fin de leçon ci-dessus.

Conclusion

Prolongements possibles

En guise de conclusion, je dirai qu'il y a encore beaucoup de travail à faire avant d'aboutir à des résultats concrets. Deux directions assez indépendantes s'offrent à nous.

D'une part, il y a une étude des concepts et fonctions didactiques relatifs au type d'objectifs pédagogiques caractérisant la classe de didacticiels que j'ai étudiée. L'objectif de cette étude serait la constitution d'un langage de description pédagogique spécifique à la classe de didacticiels considérée. Un tel travail est considérable et me parait exiger une collaboration étroite entre informaticiens et pédagogues.

D'autre part, on peut imaginer la réalisation de didacticiels dont la stratégie pédagogique serait figée et qui ne permettraient que des ajustements minimes (définition de quelques paramètres,...). Ces didacticiels permettraient de vérifier l'efficacité de la structure proposée ainsi que de tester les différents modules autres que le module pédagogique.

Ces deux approches, bien qu'indépendantes, peuvent présenter un certain caractère de complémentarité, l'expérimentation de didacticiels figés pouvant mettre en évidence tel ou tel aspect pédagogique.

Réflexion sur l'évolution du travail

Lors du choix de mon mémoire, j'envisageais de réaliser un didacticiel tel que j'en avais vu sur micro-ordinateur. Cela me paraissait a priori assez intéressant, tant du point de vue de leur réalisation que de leur utilité pratique. Mais je n'avais pas vu la complexité des concepts sous-jacents. Par exemple, je ne me posais pas de question sur l'objectif pédagogique visé ou sur la formalisation du niveau de connaissance atteint par l'élève.

Dans un second temps, suite à des lectures et à des conversations avec mon promoteur, je me suis rendu compte de la complexité de ces concepts ainsi que de leur nécessité pour espérer obtenir des résultats satisfaisants. C'est pourquoi je me suis alors dirigé vers l'étude d'une classe de didacticiels plutôt que vers l'étude d'un didacticiel particulier. Pendant un certain temps, la complexité des problèmes rencontrés m'a un peu désarçonné. Je me suis senti débordé, ne voyant plus comment arriver à une réalisation concrète.

Ce n'est que fin avril que les problèmes sont devenus plus clairs et que j'ai pu commencer à envisager l'étude spécifique d'un didacticiel et son implémentation. C'était évidemment fort tard pour espérer arriver à la réalisation concrète. Cette étude m'a cependant permis de découvrir le monde pédagogique et de mettre en évidence les points essentiels à l'élaboration d'un didacticiel qui se veut pédagogique.

Mon travail devrait permettre d'entrer plus rapidement dans le monde pédagogique, d'aborder ainsi l'étude spécifique plus tôt et de parvenir jusqu'à la conception d'un didacticiel opérationnel.

Annexe : Quelques exemples de règles sans méta-règle

Les règles ci-dessous ne font pas intervenir de méta-règles. Par contre, elles essayent de reproduire un peu la façon de penser d'un être humain, c'est-à-dire qu'il y a des règles redondantes, des règles qui sont des cas particuliers d'autres. Ces redondances sont cependant assez limitées et devraient être beaucoup plus nombreuses. De plus, ces règles ne font pas intervenir la soustraction.

Les règles sont séparées en deux classes :

- celles qui simplifient l'expression, par exemple en divisant le numérateur et le dénominateur d'une fraction par un diviseur commun
- celles qui effectuent des opérations arithmétiques, transformant par exemple div(mul(2,3),5) en div(6,5).

Règles de simplification

$\frac{a}{b} = \frac{c}{d}$ où $a = c * m$ et $b = d * m$

```
simplifie(1,div(N1,D1),div(N2,D2),[N1,D1]) :-
    map(integer,[N1,D1]),pgcd(N1,D1,P),
    divise(N1,D1,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    N2 is N1/Z,D2 is D1/Z.
simplifie(2,div(N1,D1),X,[N1,D1]) :-
    pgcd(N1,D1,P),P>1,
    N2 is N1/P,D2 is D1/P,
    enleve(div(N2,D2),X).
```

$\frac{a_1 * a}{b} = \frac{c * a}{d}$ où $a_1 = c * m$ et $b = d * m$

```
simplifie(3,div(mul(N1,N2),Z1),div(mul(N3,N2),Z2),[N1,Z1]) :-
    integer(N2),pgcd(N1,Z1,P),
    divise(N1,Z1,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    N3 is N1/Z,Z2 is Z1/Z.
simplifie(4,div(mul(N1,N2),Z1),div(mul(N1,N3),Z2),[N2,Z1]) :-
    integer(N1),pgcd(N2,Z1,P),
    divise(N2,Z1,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    N3 is N2/Z,Z2 is Z1/Z.
simplifie(5,div(mul(N1,N2),Z1),X,[N1,Z1]) :-
    integer(N2),pgcd(N1,Z1,P),P>1,
    N3 is N1/P,Z2 is Z1/P,
    enleve(div(mul(N3,N2),Z2),X).
simplifie(6,div(mul(N1,N2),Z1),X,[N2,Z1]) :-
    integer(N1),pgcd(N2,Z1,P),P>1,
    N3 is N2/P,Z2 is Z1/P,
    enleve(div(mul(N1,N3),Z2),X).
```

$\frac{a}{b_1 * b} = \frac{c}{d * b}$ où $a = c * m$ et $b_1 = d * m$

```
simplifie(7,div(Z1,mul(N1,N2)),div(Z2,mul(N3,N2)),[Z1,N1]) :-
    integer(N2),pgcd(N1,Z1,P),
    divise(N1,Z1,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    N3 is N1/Z,Z2 is Z1/Z.
simplifie(8,div(Z1,mul(N1,N2)),div(Z2,mul(N1,N3)),[Z1,N2]) :-
    integer(N1),pgcd(N2,Z1,P),
    divise(N2,Z1,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    N3 is N2/Z,Z2 is Z1/Z.
simplifie(9,div(Z1,mul(N1,N2)),X,[Z1,N1]) :-
    integer(N2),pgcd(N1,Z1,P),P>1,
    N3 is N1/P,Z2 is Z1/P,
    enleve(div(Z2,mul(N3,N2)),X).
```

```
simplifie(10,div(Z1,mul(N1,N2)),X,[Z1,N2]) :-
    integer(N1),pgcd(N2,Z1,P),P>1,
    N3 is N2/P,Z2 is Z1/P,
    enleve(div(Z2,mul(N1,N3)),X).
```

$\frac{a_1 * a}{b_1 * b} = \frac{c * a}{d * b}$ où $a_1 = c * m$ et $b_1 = d * m$

```
simplifie(11,div(mul(N1,N2),mul(D1,D2)),div(mul(N3,N2),mul(D3,D2)),[N1,D1]) :-
    map(integer,[N2,D2]),pgcd(N1,D1,P),
    divise(N1,D1,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    N3 is N1/Z,D3 is D1/Z.
simplifie(12,div(mul(N1,N2),mul(D1,D2)),div(mul(N3,N2),mul(D1,D3)),[N1,D2]) :-
    map(integer,[N2,D1]),pgcd(N1,D2,P),
    divise(N1,D2,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    N3 is N1/Z,D3 is D2/Z.
simplifie(13,div(mul(N1,N2),mul(D1,D2)),div(mul(N1,N3),mul(D3,D2)),[N2,D1]) :-
    map(integer,[N1,D2]),pgcd(N2,D1,P),
    divise(N2,D1,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    N3 is N2/Z,D3 is D1/Z.
simplifie(14,div(mul(N1,N2),mul(D1,D2)),div(mul(N1,N3),mul(D1,D3)),[N2,D2]) :-
    map(integer,[N1,D1]),pgcd(N2,D2,P),
    divise(N2,D2,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    N3 is N2/Z,D3 is D2/Z.
simplifie(15,div(mul(N1,N2),mul(D1,D2)),X,[N1,D1]) :-
    map(integer,[N2,D2]),pgcd(N1,D1,P),P>1,
    N3 is N1/P,D3 is D1/P,
    enleve(div(mul(N3,N2),mul(D3,D2)),X).
simplifie(16,div(mul(N1,N2),mul(D1,D2)),X,[N1,D2]) :-
    map(integer,[N2,D1]),pgcd(N1,D2,P),P>1,
    N3 is N1/P,D3 is D2/P,
    enleve(div(mul(N3,N2),mul(D1,D3)),X).
simplifie(17,div(mul(N1,N2),mul(D1,D2)),X,[N2,D1]) :-
    map(integer,[N1,D2]),pgcd(N2,D1,P),P>1,
    N3 is N2/P,D3 is D1/P,
    enleve(div(mul(N1,N3),mul(D3,D2)),X).
```

```
simplifie(18,div(mul(N1,N2),mul(D1,D2)),X,[N2,D2]) :-
    map(integer,[N1,D1]),pgcd(N2,D2,P),P>1,
    N3 is N2/P,D3 is D2/P,
    enleve(div(mul(N1,N3),mul(D1,D3)),X).
```

$\frac{a}{b} * c = \frac{a}{d} * e$ où $c = e * m$ et $b = d * m$

```
simplifie(19,mul(div(N,D1),Z1),mul(div(N,D2),Z2),[D1,Z1]) :-
    integer(N),pgcd(D1,Z1,P),
    divise(D1,Z1,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    D2 is D1/Z,Z2 is Z1/Z.
simplifie(20,mul(Z1,div(N,D1)),mul(Z2,div(N,D2)),[D1,Z1]) :-
    integer(N),pgcd(D1,Z1,P),
    divise(D1,Z1,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    D2 is D1/Z,Z2 is Z1/Z.
simplifie(21,mul(div(N,D1),Z1),X,[D1,Z1]) :-
    integer(N),pgcd(D1,Z1,P),P>1,
    D2 is D1/P,Z2 is Z1/P,
    enleve(mul(div(N,D2),Z2),X).
simplifie(22,mul(Z1,div(N,D1)),X,[D1,Z1]) :-
    integer(N),pgcd(D1,Z1,P),P>1,
    D2 is D1/P,Z2 is Z1/P,
    enleve(mul(Z2,div(N,D2)),X).
```

$\frac{a}{b_1} * \frac{a_1}{b} = \frac{a}{d} * \frac{c}{b}$ où $a_1 = c * m$ et $b_1 = d * m$

```
simplifie(23,mul(div(N,D1),div(N1,D)),mul(div(N,D2),div(N2,D)),[N,D1,N1,D]) :-
    map(integer,[N,D]),pgcd(D1,N1,P),
    divise(D1,N1,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    D2 is D1/Z,N2 is N1/Z.
simplifie(24,mul(div(N1,D),div(N,D1)),mul(div(N2,D),div(N,D2)),[N1,D,N,D1]) :-
    map(integer,[N,D]),pgcd(D1,N1,P),
    divise(D1,N1,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    D2 is D1/Z,N2 is N1/Z.
```

```
simplifie(25,mul(div(N,D1),div(N1,D)),X,[N,D1,N1,D]) :-
    map(integer,[N,D]),pgcd(D1,N1,P),P>1,
    D2 is D1/P,N2 is N1/P,
    enleve(mul(div(N,D2),div(N2,D)),X).
simplifie(26,mul(div(N1,D),div(N,D1)),X,[N1,D,N,D1]) :-
    map(integer,[N,D]),pgcd(D1,N1,P),P>1,
    D2 is D1/P,N2 is N1/P,
    enleve(mul(div(N2,D),div(N,D2)),X).

/* Procédure enlevant les 1 superflus d'une expression */
enleve(X,X) :- integer(X).
enleve(div(X,Y),Z) :- enleve(Y,Y2),Y2=1,!,enleve(X,Z).
enleve(div(X,Y),div(X2,Y2)) :- enleve(X,X2),enleve(Y,Y2).
enleve(mul(X,Y),Z) :- enleve(X,X2),X2=1,!,enleve(Y,Z).
enleve(mul(X,Y),Z) :- enleve(Y,Y2),Y2=1,!,enleve(X,Z).
enleve(mul(X,Y),mul(X2,Y2)) :- enleve(X,X2),enleve(Y,Y2).
enleve(add(X,Y),add(X2,Y2)) :- enleve(X,X2),enleve(Y,Y2).
```

Règles effectuant des opérations arithmétiques

```
effectue(51,mul(X,Y),Z,[X,Y]) :-
    map(integer,[X,Y]),Z is X*Y.
effectue(52,add(X,Y),Z,[X,Y]) :-
    map(integer,[X,Y]),Z is X+Y.
```

$$\frac{\frac{a}{b}}{c} = \frac{a}{b*c}$$

```
effectue(53,div(div(N,D),Z),div(N,mul(D,Z)),[N,D,Z]) :-
    map(integer,[N,D,Z]).
```

$$\frac{\frac{a}{b}}{c} = \frac{a*c}{b}$$

```
effectue(54,div(Z,div(N,D)),div(mul(Z,D),N),[N,D,Z]) :-
    map(integer,[N,D,Z]).
```

$$\frac{\frac{a}{b}}{\frac{c}{d}} = \frac{a*d}{b*c}$$

```
effectue(55,div(div(N1,D1),div(N2,D2)),mul(div(N1,D1),div(D2,N2)),
                                                    [N1,D1,N2,D2]) :-
    map(integer,[N1,D1,N2,D2]).
```

$$\frac{a}{b} * c = \frac{a*c}{b}$$

```
effectue(56,mul(div(N,D),Z),div(mul(N,Z),D),[N,D,Z]) :-
    map(integer,[N,D,Z]).
effectue(57,mul(Z,div(N,D)),div(mul(Z,N),D),[N,D,Z]) :-
    map(integer,[N,D,Z]).
```

$$\frac{a}{b} * \frac{c}{d} = \frac{a*c}{b*d}$$

```
effectue(58,mul(div(N1,D1),div(N2,D2)),div(mul(N1,N2),mul(D1,D2)),
                                                    [N1,D1,N2,D2]) :-
    map(integer,[N1,D1,N2,D2]).
```

$$\frac{a}{b}+c=\frac{a+c*b}{b}$$

```
effectue(59,add(div(N,D),Z),div(add(N,mul(Z,D)),D),[N,D,Z]) :-
    map(integer,[N,D,Z]).
effectue(60,add(Z,div(N,D)),div(add(mul(Z,D),N),D),[N,D,Z]) :-
    map(integer,[N,D,Z]).
```

$$\frac{a}{b}+\frac{c}{b}=\frac{a+c}{b}$$

```
effectue(61,add(div(N1,D),div(N2,D)),div(add(N1,N2),D),[N1,N2,D]) :-
    map(integer,[N1,N2,D]).
```

$$\frac{a}{b_1}+\frac{c}{b_2}=\frac{a*d_1+c*d_2}{b} \text{ où } b=b_1*d_1=b_2*d_2$$

```
effectue(62,add(div(N1,D1),div(N2,D2)),div(add(mul(N1,D4),mul(N2,D3)),D5),
                                                    [N1,D1,N2,D2]) :-
    map(integer,[N1,N2]),pgcd(D1,D2,P),
    divise(D1,D2,2,P,Z),on(Z,[2,3,4,5,7,10,100]),
    D3 is D1/Z,D4 is D2/Z,D5 is D3*D4*Z.
effectue(63,add(div(N1,D1),div(N2,D2)),div(add(mul(N1,D4),mul(N2,D3)),D5),
                                                    [N1,D1,N2,D2]) :-
    map(integer,[N1,N2]),pgcd(D1,D2,P),P>1,
    D3 is D1/P,D4 is D2/P,D5 is D3*D4*P.
```

$$\frac{a}{b}+\frac{c}{d}=\frac{a*d+c*b}{b*d}$$

```
effectue(64,add(div(N1,D1),div(N2,D2)),
                div(add(mul(N1,D2),mul(N2,D1)),mul(D1,D2)),[N1,D1,N2,D2]) :-
    map(integer,[N1,D1,N2,D2]).
```

Bibliographie

Derek Sleeman and J. S. Brown :
Intelligent Tutoring Systems
Academic Press (London), 1983

Dick Bierman, Joost Breuker and Jacobijn Sandberg :
Artificial Intelligence and Education
IOS (Amsterdam), 1989

W.F. Clocksin and C.S. Mellish :
Programming in Prolog
Springer-Verlag Berlin Heidelberg (New-York), 1981

Leon Sterling and Ehud Shapiro :
The Art of Prolog
The MIT Press (Cambridge, Massachusetts), 1986

Nicky Johns :
MacProlog Reference Manual
Logic Programming Associates Ltd, 1990